Finalised dependability framework and evaluation results

Antonia Bertolino, Antonello Calabro, Silvano Chiaradonna, Felicita Di Giandomenico, Antinisca Di Marco, Valerie Issarny, Massimiliano Itria, Francesca Lonetti, Marta Kwiatkowska, Fabio Martinelli, et al.

CONNECT

Emergent Connectors for

Eternal Software Intensive Networked Systems

ICT FET IP Project

Deliverable D5.4

Finalised dependability framework and evaluation results

SEVENTH FRAMEWORK PROGRAMME

http://www.connect-forever.eu

Inria · DOCOMO Euro-Labs · Ambientic · Lancaster University · Thales · Università dell'Aquila · technische universität dortmund · University of Oxford · Uppsala Universitet

Project Number	:	231167
Project Title	:	CONNECT – Emergent Connectors for Eternal Software Intensive Networked Systems
Deliverable Type	:	Report, Prototype

Deliverable Number	:	D5.4
Title of Deliverable	:	Finalised dependability framework and evaluation results
Nature of Deliverable	:	Report, Prototype
Dissemination Level	:	PU
Internal Version Number	:	1.0
Contractual Delivery Date	:	30 November 2012
Actual Delivery Date	:	10 January 2012
Contributing WPs	:	WP5
Editor(s)	:	Antonia Bertolino
Author(s)	:	Antonia Bertolino, Antonello Calabró, Silvano Chiaradonna, Felicita Di Giandomenico, Antinisca Di Marco, Valerie Issarny, Massimiliano Itria, Francesca Lonetti, Marta Kwiatkowska, Fabio Martinelli, Ilaria Matteucci, Charles Morisset, Nicola Nostro, Hongyang Qu, Alberto Ribolini, Anna Vaccarelli,
Reviewer(s)	:	Paul Grace, Valerie Issarny

Abstract

The ambitious aim of CONNECT *is to achieve universal interoperability between heterogeneous Networked Systems by means of on-the-fly synthesis of the* CONNECT*ors through which they communicate. The goal of WP5 within* CONNECT *is to ensure that the non-functional properties required at each side of the connection going to be established are fulfilled, including dependability, performance, security and trust, or, in one overarching term,* CONNECT*ability. To model such properties, we have introduced the CPMM meta-model which establishes the relevant concepts and their relations, and also includes a Complex Event language to express the behaviour associated with the specified properties. Along the four years of project duration, we have developed approaches for assuring* CONNECT*ability both at synthesis time and at run-time. Within* CONNECT *architecture, these approaches are supported via the following enablers:*

- *the Dependability and Performance analysis Enabler, which is implemented in a modular architecture supporting stochastic verification and state-based analysis. Dependability and performance analysis also relies on approaches for incremental verification to adjust* CONNECT*or parameters at run-time;*
- *the Security Enabler, which implements a Security-by-Contract-with-Trust framework to guarantee the expected security policies and enforce them accordingly to the level of trust;*
- *the Trust Manager that implements a model-based approach to mediate between different trust models and ensure interoperable trust management.*

The enablers have been integrated within the CONNECT *architecture, and in particular can interact with the* CONNECT *event-based monitoring enabler (*GLIMPSE *Enabler released within WP4) for run-time analysis and verification. To support a Model-driven approach in the interaction with the monitor, we have developed a CPMM editor and a translator from CPMM to the* GLIMPSE *native language (Drools). In this document that is the final deliverable from WP5 we first present the latest advances in the fourth year concerning CPMM, Dependability&Performance Analysis, Incremental Verification and Security. Then, we make an overall summary of main achievements for the whole project lifecycle. In appendix we also include some relevant articles specifically focussing on* CONNECT*ability that have been prepared in the last period.*

Keyword List

CONNECT*ability,* CONNECT*ed System,* CONNECT*or, Dependability, Enabler, Monitoring, Networked System, Non-functional Properties, Performance, Property Meta-model, Security, Security Policy, Security-by-Contract, State-Based Stochastic Methods, Stochastic Model Checking, Trust*

Document History

Version	Type of Change	Author(s)
0.1	Proposed template and TOC	A. Bertolino
0.2	First contributions to chapters by all contributors	All
0.3	Complete draft for internal review	All
0.4	Revision after internal review by Grace	A. Bertolino, F. Di Giandomenico, F. Lonetti
0.5	Revision after internal review by Issarny	All
1.0	Final version	A. Bertolino

Document Review

Date	Version	Reviewer	Comment
23/11/2012	V 0.3	P. Grace - Lanc	Complete review with several specific comments
05/12/2012	V 0.5	V. Issarny - Inria	Review throughout with mostly comments to improve style

Table of Contents

List of Figures

List of Tables

1 Introduction

CONNECT Workpackage 5 (WP5) on "Dependability Assurance" encompasses dependability and performance analysis and verification, as well as security, privacy and trust management within the CONNECTed world; in a word, the role of WP5 is to ensure that a CONNECTed system yields the required CONNECTability properties. More precisely, CONNECTability refers to the ability of CONNECT enablers to provide a CONNECTed system with the intended properties of Dependability, Performance, Security and Trust in justifiable way.

1.1 WP5 Tasks

We recall from the DoW that WP5 activity is structured into the following four tasks:

- **Task 5.1. Dependability metrics for open dynamic systems:** the aim is to revisit classical dependability metrics to account for dynamic CONNECTions in open, evolutionary networks and specify relevant properties to be ensured.
- **Task 5.2. Dependability verification & validation (V&V) in evolving, adaptive contexts:** this aims at developing approaches for quantitative verification of dependability and performance properties, and for lightweight adaptive monitoring that is meant to detect at run-time potential problems and to provide feedback to learning and synthesis activities.
- **Task 5.3. Security and privacy:** this aims at adapting and extending existing techniques for security-by-contract checking and enforcement.
- **Task 5.4. Distributed trust management:** this aims at developing a unifying theory for trust and a corresponding reputation scheme.

1.2 WP5 Results

Concerning Task 5.1, we have defined the CONNECT property meta-model (CPMM) that supports a model-driven approach to the specification of CONNECTability properties. CPMM is used as the exchange language among CONNECT Enablers to communicate and manage non-functional properties: thus properties defined according to such meta-model can be used, for example, as input for the dependability&performance Enabler to verify specified CONNECTability properties, or as instrumentation for the monitoring Enabler to generate suitable probes to monitor the specified properties on the CONNECTors, and so on.

Concerning Task 5.2, we defined the architecture of the Dependability&Performance (DEPER) Enabler. Before deployment, DEPER cooperates with the Synthesis Enabler to verify whether the dependability and performance requirements requested by the NSs can be satisfied by the CONNECTor being synthesised. At run-time, DEPER cooperates with the Monitoring Enabler to check that the assumptions on which the analysis is based remain valid. DEPER supports both stochastic model checking and state-based stochastic evaluation approaches, using two analysis engines based on Möbius and on Prism, respectively. We have also introduced incremental verification, which allows for refining the analysis after changes to the system, without having to redo it all from scratch.

With regard to Task 5.3, we presented the Security-by-Contract-with-Trust (SxCxT) infrastructure, which refined the existing security-by-contract approach through a detailed study of the CONNECT-specific security threat models, and showed how it can verify that the security contract and the required trust levels are satisfied, or otherwise how it can enforce them at run-time. We have also introduced an approach to negotiate credential-based trust access levels.

Concerning Task 5.4, this has been partially covered by the SxCxT approach above outlined. In addition, we have introduced a generic trust meta-model as a basis to express and compose a wide range of trust models, so that heterogeneous trust management systems belonging to different NSs can interoperate transparently. To this aim, novel mediation algorithms have been developed to overcome

the heterogeneity between the trust metrics, relations and operations associated with the composed trust models.

The WP has released several enablers, as well as novel algorithms and approaches, for performing analysis during synthesis, and run-time validation after CONNECTor deployment. Overall, the workpackage has developed a comprehensive and practical framework that is well integrated within the CONNECT architecture, and supports the novel vision of on-the-fly CONNECTion pursued in the project by providing enablers for assessing and ensuring the appropriate levels of dependability, performance, security and trust.

1.3 This deliverable

This is the final deliverable of WP5: we report the progress made in the last reporting period (Y4) as well as an overall summary of the key Scientific and Technological results achieved along the four years of project life. Precisely, in the next chapter we give a brief description of the latest advances in the active tasks, taking into account the M36 review comments and including the definition of the property meta-model, the analysis and incremental verification of dependability and performance properties and the enforcement of security policies. In Chapter 3 we highlight the main achievements obtained all along the project as previously reported in Deliverables D5.1, D5.2, D5.3 and in this document. In Chapter 4, we make a qualitative assessment of the achieved results against the objectives for evaluation established in D6.3. In Chapter 5 we draw conclusions and hint at future work, behind CONNECT duration.

Finally, in Appendix we include the following journal articles or book chapters that have been recently produced to disseminate CONNECT results:

P1 The paper “Meta-Modeling of Non-Functional-Properties” presents in detail our generic and comprehensive Property Meta-Model for defining non-functional properties, metrics and complex events. It shows the application of CPMM to two different domains that are synthesis and monitoring of non-functional properties and presents the results of a systematic survey of the literature on property metamodels and complex events languages.

P2 The paper “An approach to adaptive dependability assessment in dynamic and evolving connected systems” gives an overview of the interaction between the DePer Enabler and the GLIMPSE monitor.

P3 The paper “On-the-Fly Dependable Mediation between Heterogeneous Networked Systems” shows how we integrate synthesis of CONNECTors, stochastic model-based analysis performed at design time and run-time monitoring.

P4 The paper “Enhanced Connectors Synthesis to address Functional, Performance, and Dependability aspects” is an enhancement of P3 and focuses on the CONNECTor adaptation process, related to the performance and dependability mechanisms, spanning pre-deployment time and run-time.

P5 The paper “Networks of interoperable systems: an approach to enhance dependability and performance properties” extends our model-based analysis framework for the synthesis of dependable CONNECTors, by identifying generic automated strategies for the selection of dependability mechanisms bringing the highest benefits. A case study based on a global monitoring system for environment and security (GMES) is also included.

2 Y4 Achievements

In this chapter we report about the activity carried out in WP5 in the last reporting period (M37-M46). More precisely, we first address in Section 2.1 reviewers' comments at the last review (covering D5.3) and then give a short compendium of the latest advances and refinement on the CPMM, dependability and performance, incremental verification, and security, in Sections 2.2, 2.3, 2.4, 2.5, respectively.

2.1 Review Comments

In the following, we report relevant text from M36 review concerning WP5 and below each comment we include our answers.

[Point 7 p. 3/13] *While generally WP5 has made very good progress, the run-time management and monitoring of non-functional properties and the reaction to their violations remain mostly open questions. This is an area of high research relevance and potential. The current solution is incomplete and immature. It is recommended that the non-functional concerns should be handled with high priority during the remaining time. In doing so, the project should consider carefully the existing related work and build on proven results, where possible. See Section 2.b for further details.*

We fully agree that handling non-functional properties is a very important research area, but it is also a quite complex challenge. Effort has been devoted in Y4 in making the monitor interact with the other enablers in the CONNECT architecture, as described in D1.4 as well as in D4.4. Existing related work has been taken into account to the best of our knowledge and exploited for each of the various proposed solutions, including the CPMM, the dependability and security enablers as well as the GLIMPSE monitoring framework (from WP4). The original contribution of the approaches we propose concerning the run-time management and monitoring of non-functional properties stays mainly in the implementation of a model-driven solution to run-time monitoring, starting from a PMM-compliant model down to its automated translation into Drools rules that are given in input to GLIMPSE.

We would like to remark that the focus of WP5 research is in enabling CONNECTability when connecting NSs via on-the-fly synthesis of CONNECTors. In accordance to the DOW, the solutions we have proposed and developed have thus spanned both pre-deployment time and run-time management of non-functional properties, hence the time and effort we could devote to the solution relative to run-time monitoring is only a portion of the whole WP5 effort. We however intend to continue research in run-time monitoring of non-functional properties of dynamically connected system also after project closure.

[Section 2.b - Subsection: Workpackage 5 (Dependability assurance) p.6-713] This is a very ambitious (perhaps too ambitious) WP where a number of issues remain to be clarified:

a) *Constraint language*

- *Semantics: What is the distance between A and B if it is not explicitly mentioned whether start or finishing time points are meant?*

  Each event occurrence refers to an "Interval" that represents the time range in which the event occurs. Each "Interval" has two required/mandatory "Timestamps" ("begin" and "end") indicating its starting and ending date respectively. These Timestamps are equal in case of atomic/instantaneous event occurrences. For most of the CPMM operators involving two events (A and B), a "maxDistance" parameter is defined that is mandatory. This parameter represents the maximum distance between the start or end timestamp of the current event and the start or end timestamp of the correlated event, according to the semantics of the specific operator.
- *Expressivity: It seems not possible to express that a sequence of events (possibly occurrences of the same event) are executed in a pipelined (or stacked) fashion. Is this never of importance in the encountered examples? Why is it important to require "overlapped" execution in that case?*

  Many operators have been introduced in CPMM for combining primitive and composite events. Specifically, the "Seq" operator allows for expressing a sequence of occurrences of the same

event type, whereas using the “After” and “Before” operators it is possible to express a sequence of events having different type. In the latter case, for defining events executed in a pipelined fashion the “maxDistance” and “minDistance” parameters are set to zero. Taking as a reference the Drools operators [1], we also introduced in CPMM the “Overlaps” and “OverlappedBy” operators with the same expressivity of the Drools ones.

– *Relevance of “class type” attribute distinguishing performance, security, . . . : Will any given property be interpreted differently whether it is tagged security or performance? If not, why this attribute?*

  The “ClassType” attribute has two main objectives: from one side it allows for defining OCL constraints on the “workload” and “intervalTime” attributes (specifically the former is mandatory for PERFORMANCE properties while the latter is mandatory for DEPENDABILITY ones); from the other side, the “ClassType” attribute specified in the input model provides the Model2Code Transformer with useful information for distinguishing about the appropriate template that needs to be called for processing the specific “ClassType” property model.

– *Relevance of “property nature” attribute: The distinction between a “prescriptive” and a “descriptive” use of a property is intuitively clear, but it seems strange to attach this characteristic to the property alone. This does not change the characteristic of the property but its use within a given connector or service. In terms of UML concepts, it would probably be better to separately enumerate “required” or “descriptive” properties of a given system or to use corresponding roles in the relation linking properties to systems (connectors).*

  Of course, the same property can be defined as “prescriptive” for a service (connector) and “descriptive” for another one. However, attaching the “PropertyNature” attribute to the input property model is useful for the Synthesis Enabler to guide the synthesis of a new CONNECTor instance or the selection process in case suitable CONNECTors are already available.

b) *Coping with failures: The arbitrary application of standard mechanisms (such as retry, voting, error correction, etc.) in the case that monitoring shows a non-satisfactorily handling of a required non-functional property seems too naive. In general such mechanisms depend very much on the general constraints imposed by different application domains. For example, “retry” may make a lot of sense for control-oriented service requests in the context of a non-reliable communication medium, but when too many image frames or periodically measured sensor values are lost, or when the communication failures are only of type “burst”, the use of “retry” is likely not to be of any help. Similarly “voting” is generally applied to values provided by more or less reliable sensors but otherwise this mechanism is mostly of little interest.*

   The reviewers are right in pointing out the need to better address the application of dependability mechanisms for handling the non-satisfaction of required non-functional properties. Deliverable D5.3 included a preliminary discussion of involved issues and at the time of writing it, work was still ongoing. In the reporting period, we have focused more closely on defining general criteria to make the choice of an appropriate dependability mechanism, from among an available set, when necessary. As we explain in Section 2.3.3, the approach we are developing is based on the identification of aspects involved in the dependability mechanism selection (namely application constraints, fault assumptions and dependability and performance indicators) and in matching the available mechanisms with these characterizing aspects. Detailed description of the general selection method is included in the paper P5 in Appendix to this deliverable.

c) *Monitoring of non-functional properties: A general question concerns the feasibility and maturity of the approach relying on intensive monitoring of non-functional properties. For example, it is well known that monitoring of non-functional properties may interfere with non-functional properties. Likewise, monitoring (and even more enforcement) requires components (not just the connectors which are under the control of CONNECT) to be collaborative in order to provide the required QoS information. In its current form the monitoring subsystem appears as a rather immature add-on. It would certainly be useful if the project would reconsider the whole set of issues related to the monitoring of non-functional properties and the reactive adaptations. There is plenty of work on monitoring in the realms of adaptive software systems and network and system management, which may be*

exploited here. As stated before, presenting an effective and efficient solution for the management of non-functional properties with on-the-fly connectors would make a very strong contribution to the state of the art.

We acknowledge that plenty of work exists on monitoring that we can exploit here. Our solution builds on existing frameworks for monitoring, including Drools [1], HiFi [32], TESLA [25], Snoop [15], GEM [37], and other related languages and frameworks. In performing the systematic survey on meta-modeling of non-functional properties (see Paper P1 in Appendix), we also made a further investigation of newer work that drove our refinements of Y4. The originality of our contribution in CONNECT stays in two aspects: on the one side, integrating the monitoring framework within the specific CONNECT approach (hence we considered CONNECT assumptions and conventions), and on the other side, in making the monitor flexible and model-driven so that we can specify the non-functional properties to be monitored in a high-level language and have an automated translation into the monitor formalism. With reference to the specific comment that "monitoring (and even more enforcement) requires components (not just the connectors which are under the control of CONNECT) to be collaborative in order to provide the required QoS information", we also clarify that not requiring the NSs to be collaborative has been a starting assumption of our approach, and in fact we only monitor the functioning of the CONNECTed system by exclusive means of probes inserted into the CONNECTor. This might make the approach less powerful on the one side, but also less intrusive on the other.

2.2 CONNECT Property Meta-Model refinements

In the third year, we mainly focused on enhancing and refining the CONNECT Property Meta-Model (CPMM) to better specify properties and complex events. In [21] we presented a mapping of CPMM *EventType* model operators with the corresponding ones in GEM [37] and Drools Fusion [1]. A significant activity started in the previous year was the implementation, using Acceleo, of the Model-To-Code (Model2Code) transformation that translates models conforming to CPMM to Drools rules [1], which are used to configure the GLIMPSE monitoring infrastructure (GLIMPSE has been presented in Deliverable D4.2 [18]). The aim of such transformation is to allow for the dynamic configuration of the input to GLIMPSE whenever a new property to be monitored is specified and introduced in CONNECT.

During the last reporting period, work on CPMM has progressed in three different directions:

i) we further refined and improved CPMM and provided new models of properties defined in CONNECT as described in the following;

ii) we refined and completed the Model2Code transformation (we show details in Section 2.2.1);

iii) we concentrated on an extensive assessment of the comprehensiveness and flexibility of CPMM with respect to similar approaches. To this aim we performed a systematic survey of the literature on property meta-models and complex events languages [9] that we summarize in Section 2.2.2.

The main improvement of CPMM introduced in this last year is the definition of the *Action* entity (this refinement has been triggered by some reviewers comments). In the new version of CPMM (we describe it in detail in [9]), the `Property` can have an associated *Action* representing the execution of a behavior or operation of the system. The *Action* entity models an atomic and black box system behavior without capturing its complexity or details. An Action has the ActionType attribute that refers to the type of the action and can have the following values: *INTERNAL*, *INPUT* and *OUTPUT*.

We used CPMM and its complex events specification for defining new property models for the Terrorist Alert Scenario demonstration scenario [20] of CONNECT. Specifically, concerning the privacy property saying that *the photo can only be received by authorized devices*, defined in [19], we modeled the events to be observed in the communication among parties. These events are defined by the *contract* EventType that we present in detail in [11]. Through this model we show the higher expressiveness of CPMM event specification language with respect to that of GEM and Drools, as well as the benefits of the combination of the GEM and Drools features and of the extension we introduce.

2.2.1 Model2Code transformer description

We need concretely to instruct GLIMPSE about what raw data (events) to collect and how to infer whether or not a desired property is fulfilled. To address such issues, in CONNECT we provide a model-based approach to automatically convert CPMM metrics and properties specifications into a concrete monitoring setup. Precisely, the editor provided along with CPMM allows the software developer to specify a new property as a model that is conforming to the CPMM meta-model. If this model represents a property to be monitored, it is passed as input to the Model2Code Transformer that produces Drools Fusion code.

We recall that the Model2Code Transformer component has been implemented using the Acceleo code generator IDE [2] that supports the automated code generation from UML and EMF and uses the *template*s to generate code (or text) from a model. The implemented Model2Code Transformer has a *.mtl* extension, it takes as inputs the CPMM ecore models (specifically Core.ecore, EventSet.ecore, EventType.ecore, Metrics.ecore, MetricsTemplate.ecore, Property.ecore) and the model (compliant to CPMM) of the property, metrics or event to be monitored, and produces one or more Drools rules, that are provided in input to the GLIMPSE complex event processor.

The Model2Code Transformer we developed consists of a main generation rule or template that calls a certain number of other templates, each one is applied on a CPMM model element (or object) to produce some text. In each template there are static areas, they will be included in the generated file as they are defined, and dynamic areas that correspond to the expression evaluation on the current object.

Figure 2.1 shows the structure of the transformer code and the relations among its templates. We provide here the key concepts of the transformer, showing some templates and refer to [41] for the overall Model2Code implementation.

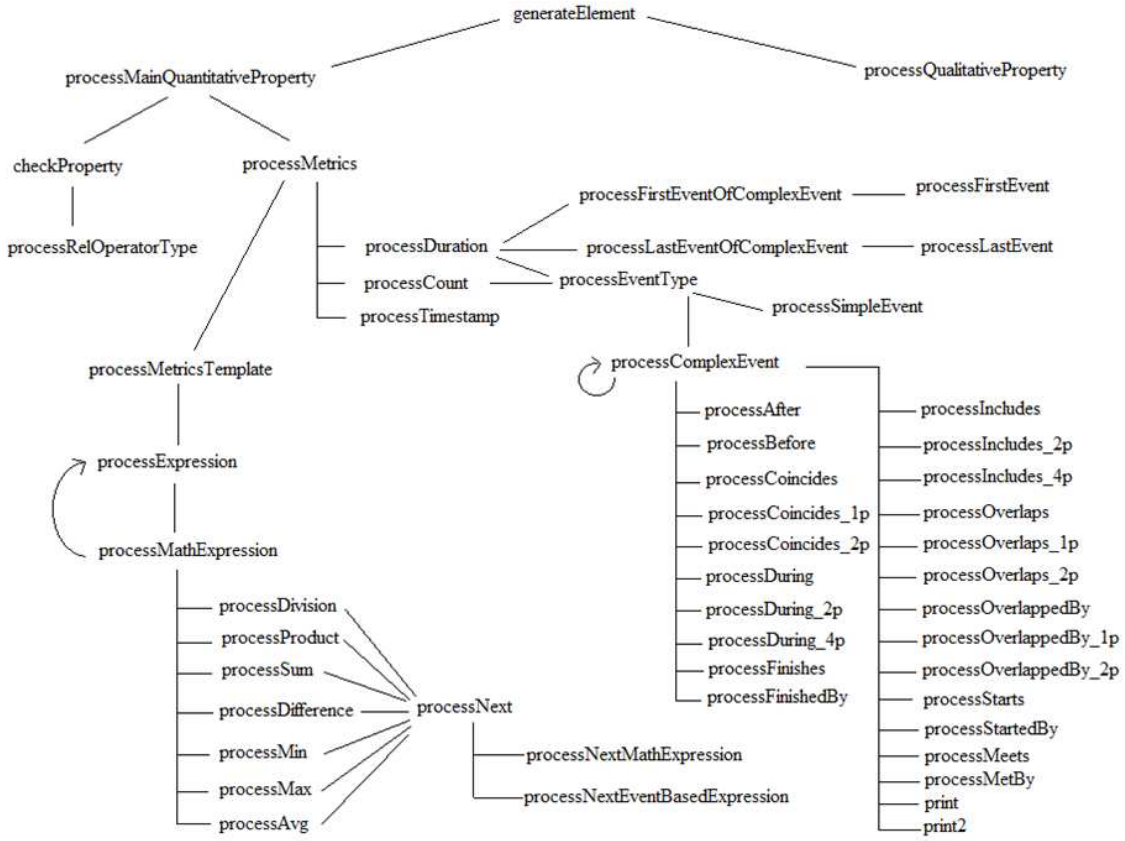


Figure 2.1: Transformer templates graph

```
1  [comment encoding = UTF-8 /]
2  [module generate('cpmm/model/Core.ecore', 'cpmm/model/EventType.ecore', 'cpmm/model/EventSet.
       ecore', 'cpmm/model/Metrics.ecore', 'cpmm/model/MetricsTemplate.ecore', 'cpmm/model/Property
       .ecore')]
3  [import cpmm::acceleo::utilities::utility]
4  [template public generateElement(p : Property)]
5  [comment @main/]
6  [file (p.name, false, 'Cp1252')]
7   [if p.oclIsTypeOf(QuantitativeProperty)]
8    [processMainQuantitativeProperty(p)/]
9   [elseif (p.oclIsTypeOf(QualitativeProperty))]
10   [processQualitativeProperty(p)/]
11  [/if]
12 [/file]
13 [/template]
```

Listing 2.1: Main Template of Model2Code Transformer

As showed in Listing 2.1, after the heading section that defines which meta-model the template will apply for and the generated file, a main template (called *generateElement*) is defined that represents the main generation rule. It takes as input the property model and specifies that if this model is about a quantitative property, then a specific template named *processMainQuantitativeProperty* is called, otherwise the *processQualitativeProperty* is called. The *processMainQuantitativeProperty* calls a specific template processing the associated metrics. In this template (see Listing 2.2) the Metrics model is navigated for identifying the associated MetricsTemplate model, then the MetricsTemplate parameters, the associated EventBasedExpressions and their operators. These EventBasedExpressions represent expressions based on events and their name represents the observable, simple or complex, event/behavior

the operator applies to. This operator has to be applied to the whole set of event occurrences of the observed EventType. We defined three different templates named processCount, processDuration and processTimestamp corresponding to the three different operator type values defined in CPMM that are CARDINALITY, DURATION and TIMESTAMP.

The three templates (processCount, processDuration and processTimestamp) take as input an EventSet and generate three different Drools rules that: count the event occurrences associated to an EventType, compute the duration of the events occurrences, compute the timestamp of the event occurrences associated to an EventType in the time window dimension specified in the EventSet, respectively.

```
1  [template public processMetrics(m : Metrics)]
2  [comment ————————————————/]
3  [processMetricsTemplate (m.metricsTemplate , m)/]
4  [comment ————————————————/]
5  [if (m.metricsTemplate.definition.oclIsTypeOf(MathExpression))]
6    [for (op : Expression | m.metricsTemplate.definition.oclAsType(MathExpression).operands)]
7      [if (op.oclIsTypeOf(MathExpression))]
8     [for (op1 : Expression | op.oclAsType(MathExpression).operands)]
9           [if (op1.oclIsKindOf(NamedExpression))]
10        [if (op1.oclAsType(NamedExpression).oclIsTypeOf(EventBasedExpression))]
11              [if (op1.oclAsType(NamedExpression).oclAsType(EventBasedExpression).operator.
                    toString() = 'CARDINALITY')]
12                [for (act : MetricsParameter | m.actualParameters)]
13                    [if (act.oclIsTypeOf(EventBasedMetricsParameter))]
14                      [for (tp: EStringToNamedExpressionObjectMap | m.metricsTemplate.
                            TemplateParameters)]
15                        [if ((tp.key = act.oclAsType(EventBasedMetricsParameter).name) and (
                              tp.value=op1))]
16                          [processCount(act.oclAsType(EventBasedMetricsParameter).eventSet)
                              /]
17                        [/if]
18                      [/for]
19                    [/if]
20                [/for]
21     [elseif (op1.oclAsType(NamedExpression).oclAsType(EventBasedExpression).operator.
            toString() = 'DURATION')]
22          [for (act : MetricsParameter | m.actualParameters)]
23                    [if (act.oclIsTypeOf(EventBasedMetricsParameter))]
24                      [for (tp: EStringToNamedExpressionObjectMap | m.metricsTemplate.
                            TemplateParameters)]
25                        [if ((tp.key = act.oclAsType(EventBasedMetricsParameter).name) and (
                              tp.value=op1))]
26                          [processDuration(act.oclAsType(EventBasedMetricsParameter).
                              eventSet)/]
27                        [/if]
28                      [/for]
29                    [/if]
30                [/for]
31     [elseif (op1.oclAsType(NamedExpression).oclAsType(EventBasedExpression).operator.toString
            () = 'TIMESTAMP')]
32           [for (act : MetricsParameter | m.actualParameters)]
33                    [if (act.oclIsTypeOf(EventBasedMetricsParameter))]
34                      [for (tp: EStringToNamedExpressionObjectMap | m.metricsTemplate.
                            TemplateParameters)]
35                        [if ((tp.key = act.oclAsType(EventBasedMetricsParameter).name) and (
                              tp.value=op1))]
36                          [processTimestamp(act.oclAsType(EventBasedMetricsParameter).
                              eventSet)/]
37                        [/if]
38                      [/for]
39                    [/if]
40                [/for]
41              [/if]
42            [/if]
43          [/if]
44        [/for]
45     [/if]
46   [/for]
```

```
47 [/if]
48 [/template]
```

Listing 2.2: ProcessMetrics Template of Model2Code Transformer

```
1  [template public processFirstEvent(ce : ComplexEvent)]
2  [if (ce.operator.oclIsTypeOf(During))]
3  [for (op : EventType | ce.composedBy)]
4    [if (not (op.oclIsTypeOf(ComplexEvent)) and (op.compositionOrder.toString() = 'FIRST'))]
5    $event_[op.name/] : [op.name/][processDuring(ce.operator.oclAsType(During))/]
6    [elseif (not (op.oclIsTypeOf(ComplexEvent)) and op.compositionOrder.toString() = 'SECOND')]
7    $event_[op.name/] : [op.name/]
8    [elseif (op.oclIsTypeOf(ComplexEvent))]
9    [processComplexEvent(op.oclAsType(ComplexEvent))/]
10   [/if]
11  [/for]
12 [elseif (ce.operator.oclIsTypeOf(Seq))]
13 [for (op : EventType | ce.composedBy)]
14     [if (not (op.oclIsTypeOf(ComplexEvent)))]
15   [if(ce.operator.oclAsType(Seq).SeqOrder.toString() = 'LAST')]
16     $event_[op.name/] : [op.name/]['('/] this [ce.operator.oclAsType(Seq).eClass().Print(ce.
         operator.oclAsType(Seq).eClass(), ce.operator.oclAsType(Seq).minLenght-1, op.name.
         toString())/][')'/]
17   [elseif(ce.operator.oclAsType(Seq).SeqOrder.toString() = 'FIRST')]
18     $event_[op.name/] : [op.name/]['('/] this [ce.operator.oclAsType(Seq).eClass().Print2(ce.
         operator.oclAsType(Seq).eClass(), ce.operator.oclAsType(Seq).minLenght-1, op.name.
         toString())/][')'/]
19   [else]
20     $event_[op.name/] : [op.name/]['('/] this [ce.operator.oclAsType(Seq).eClass().Print(ce.
         operator.oclAsType(Seq).eClass(), ce.operator.oclAsType(Seq).minLenght-1, op.name.
         toString())/][')'/]
21   [/if]
22   [else]
23   [processComplexEvent(op.oclAsType(ComplexEvent))/]
24    [/if]
25  [/for]
26 [/if]
27 [/template]
```

Listing 2.3: ProcessEventType Template of Model2Code Transformer

The three templates (processCount, processDuration and processTimestamp) call another template (processEventType) that navigates the EventType model and maps all the CPMM complex events operators into the corresponding Drools operators.

As an example we show in Listing 2.3 how to map the *During* and *Seq* complex events operators in the corresponding Drools operators. Note that the *Seq* operator is not defined in Drools, then we model it as a sequence of *after* or *before* operators by means of the Print and Print2 Java services (lines 16, 18 and 20). By Java services, Acceleo supports the user code blocks specification in some areas of the generated file. This standard Java code is executed by accessing to it from a query in the Acceleo template. We presented in [11] a complete mapping of CPMM complex events operators with those of GEM and Drools.

We used the Model2Code Transformer for generating the Drools code used to monitor two properties of interest for the Terrorist Alert Scenario of CONNECT: a dependability property (coverage property) and a performance property (latency property).

Specifically, the *coverage* property, defined in Deliverable 5.3 [21], says that: *the average percentage of guard devices that are reached in 10 seconds by the alert message must be greater than* 70%. This means that, after 10 seconds from the *EmergencyAlert*, at least 70% of guard devices reply with an *eAck*. The *eAlert_eAck EventType* required by the coverage property is presented in Figure 2.2. It is a complex event representing the *EmergencyAlert* with its related *eAck* from the guards. In particular, *eAlert_eAck* has a *Before* operator with *maxDistance* parameter equal to 10 and it is composed by *eAlert* simple *EventType* and *Seq_eAck* complex *EventType*. The latter is another *ComplexEvent* type with *Seq* operator representing the sequence of guards messages acknowledging the alert.

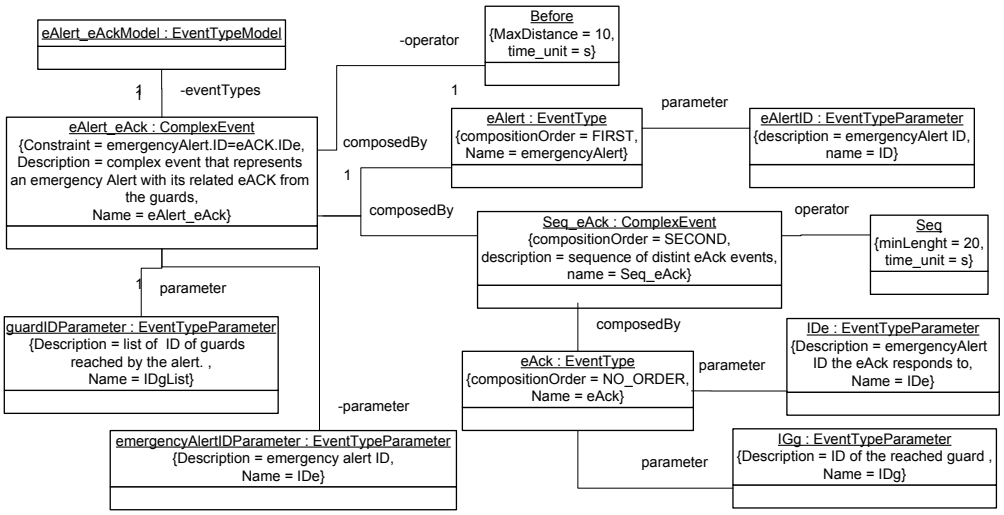


Figure 2.2: Sequence of Ack for an Alert

As an example, we show in Listing 2.4 the Drool rule derived applying the Model2Code Transformer to the *eAlert_eAck* model of Figure 2.2. This rule counts the number of *eAler_eAck* events defined for the coverage property that happen in a time window of 10 seconds and saves this information into a new generated event (called *counteAlert_eAck*). The rule uses the *before* (line 12) operator of Drools and translates the *Seq* operator of CPMM in a sequence of *after*(line 14) operators of Drools.

We refer to Deliverable 5.3 [21] for the other models of the coverage property and the corresponding Drools code whereas for the latency property models and the corresponding Drools rules we refer to [41].

```
1  declare Total_eAlert_eAckcaptured
2   @total: int
3  end
4
5  rule "Number of eAlert_eAck incomingEvents"
6  no-loop
7  salience 999
8  dialect "java"
9  when
10  $totaleAlert_eAck: Number();
11   from accumulate(
12  $event_eAlert_eAck: emergencyAlert( this before
13  $event_seq_eAck:
14  eAck( this after eAck after eAck after eAck after eAck after eAck after eAck after eAck
        after eAck after eAck after eAck after eAck after eAck after eAck after eAck after eAck
        after eAck after eAck after eAck after eAck )
15  )over window:time(10s) from entry-point "DEFAULT", count($event_eAlert_eAck))
16 then
17  Total_eAlert_eAckcaptured counteAlert_eAck = new Total_eAlert_eAckcaptured();
18  counteAlert_eAck.setTotal($totaleAlert_eAck)
19  insert(counteAlert_eAck);
20  System.out.println( "Number of Incoming events: " + $totaleAlert_eAck);
21 end
```

Listing 2.4: Drools Code generated by Model2Code Transformer for the *coverage* property

2.2.2 Systematic survey

In order to compare CPMM with existing similar approaches we performed a systematic survey on meta-models addressing non-functional properties, metrics and complex events languages [9]. Below we briefly present before the research method and then the obtained results. The survey is detailed in paper P1 in

Appendix currently submitted for publication.

Research method

This survey has been conducted following the guidelines for systematic review in software engineering research proposed by Kitchenham [34]. These guidelines cover three aspects of a systematic review: planning the review, conducting the review and reporting the results. In the planning phase we identified the goal of this review that is understanding the current state of art in modeling non-functional properties comparing different languages. Specifically, we formulated the following research questions (RQ):

(RQ1) which techniques/approaches have been used for modeling non-functional properties?

(RQ2) how similar are existing approaches to CPMM?

(RQ3) which automation level is provided by the different proposed solutions?

(RQ4) which is the purpose of the different modeling solutions?

For selecting the most appropriate papers dealing with non-functional properties specification we performed an automatic search. Similarly to the review presented in [4] concerning MDD approaches dealing with non-functional properties, we have based our search method on Web of Science (WoS). Specifically, we have searched WoS by topic (title and abstract and keywords) using Science Citation Index Expanded (SCI-EXPANDED) and Conference Proceedings Citation Index- Science (CPCI-S) citation databases and selected English papers from 1990 to 2012 (precisely 26 July 2012 is the date in which we performed the search).

According to the research goals we defined the following search string: (“model driven” OR “metamodel” OR “meta model” OR “metamodel” OR “model-driven” OR “MDE”) AND (((“non functional” OR “security” OR “dependability” OR “performance” OR “nonfunctional” OR “non functional” OR “quality” OR “NFR”) AND (“requirements” OR “concepts” OR “properties”)) OR “metric*” OR “events”).

With respect to the query in [4], we introduced relevant key words that are specific to our research domain, such as metamodel, metric, event, security, dependability, and performance. Specifically, our search string consists of three parts: a first part addressing meta-modeling proposals in MDE systems, the second part is related to non-functional properties or requirements and in particular to security, dependability and performance ones, the last part links works that describe metrics or events modeling.

From this automatic search we obtained 853 papers, which reduced to 74 after reading the title and the source, then reduced to 28 after reading the abstract and finally reduced to 23 after reading the full text.

For complementing the results of the automatic search, we selected five additional relevant works [1, 25, 29, 30, 37] that we found in a previous manual search and that were not found by the WoS-based search.

Results

Table 2.1 shows the classification framework we adopted in this study. The extracted data were classified according to seven dimensions that are: type of non-functional property, domain, instrument, expressiveness, formalization, purpose and automation. We detail each of them in the following:

Type of non-functional property and metric. It refers to the type of non-functional property and metrics that is the target of the modeling activity. All the analyzed papers focus on one or more types of non-functional properties. Most of them address security, other ones performance or dependability. Other works deal with the specification of metrics related to software product and process.

Domain. It represents the technological context in which the non-functional properties are defined and evaluated. Modeling of non-functional properties fits in various application domains. Many approaches focus on SOA applications and distributed systems, whereas existing proposals deal with database management systems, data warehouses, safety-related standard, multi-agent systems and attack modeling.

Instrument. It refers to the formalism used for expressing the non-functional properties. Many authors define UML profiles, others design a metamodel specifically conceived for expressing non-functional properties.

Expressiveness. With this concept we mean the power of the modeling solution in terms of what it is able to express, specifically properties and/or metrics and/or events.

Formalization. It represents the level of formalization used by the different modeling languages for specifying a non-functional property. This dimension can be characterised as informal, which means described as a natural language piece of text, or in a semi-formal language using for instance names and values tied by a relational operator, or by a more formal description usually represented by a mathematical formula.

Purpose. The defined non-functional property models can be aimed to support the different activities of the model-driven development process such as analysis, synthesis, monitoring and testing.

Automation. It refers to the support provided by the different solutions in terms of automated facilities for properties specification and tools for the Model2Code and/or Model2Text transformations.

Looking at the results summarised in Table 2.1, we can conclude that although valuable approaches exist for modeling non-functional properties, none of them proposes a flexible and comprehensive solution similar to that of CPMM. The main advantages of CPMM with respect to similar approaches are:

- it addresses non-functional properties spanning over performance, dependability and security. The analyzed approaches address only a subset of the properties addressed in CPMM or they target general non-functional properties without specifying the type.
- it is independent from the specific application domain. Most approaches are defined for a specific application domain and few ones are domain-independent as CPMM.
- it is more expressive than all analyzed solutions since it represents the only proposal that allows for expressing properties, metrics and events. All the other approaches are centered in only one or two of these modeling activities.
- concerning the phases of the model-driven development process, CPMM supports most of them. Indeed it can be used for monitoring and synthesis as we show in [9] and also for testing purposes as we sketch in [38]. Most of the analyzed approaches focus only on design and analysis, other as CPMM, address monitoring.

As other proposals, CPMM is a specific metamodel designed for specifying non-functional properties. It adopts a formal description for defining metrics by introducing the *MetricsTemplate* concept containing the mathematical definition of the metrics. Other proposed metamodels adopt a semi-formal or informal description. Several authors propose UML extensions (UML profile) for modeling non-functional properties, also in this case the adopted description is semi-formal or informal. Finally, concerning the automated support provided from different solutions, CPMM as other approaches, has an associated editor that allows for deriving new properties, metrics and events and similarly to other approaches, CPMM supports automated procedures for Model2Code transformation.

Table 2.1: Classification of approaches dealing with modeling of non-functional properties according to our systematic survey

Ref	Type of NFP	Domain	Instrument	Expressiveness	Formalization	Purpose	Automation
CPMM	performance, security, dependability	any	own metamodel	properties metrics, events	formal	synthesis, monitoring testing	editor, M2T transformer
Systematic Survey Results							
[7, 8]	security	security-design models for distributed systems	UML's OCL formula	properties	formal	model analysis	editor, analysis and validation tool
[42]	security	any	own metamodel	requirements	semi-formal	analysis, design implementation	editor, M2M and M2T transformer
[54]	security	active database	own metamodel	events	formal	monitoring	none
[55]	security	airborne systems based on RTCA DO-178B standard	UML profile	properties	informal	assessment certification	none
[13]	security	multi-agent systems	own metamodel	requirements	semi-formal	design, analysis	none
[39]	security	distributed systems	own metamodel	requirements, metrics	informal	attack modeling	none
[50]	security	data warehouses	UML profile	requirements	semi-formal	design	M2M and M2T transformer
[27]	any	information systems	own metamodel	quality attributes and metrics	informal	design	none
[43]	security	business process	UML activity diagram profile	requirements	informal	design, analysis	M2M
[40, 47]	security	SOA business process	UML profile	requirements	informal	design	none
[46]	any	software products process models	own metamodel	metrics	formal	design specification model transformations	none
[52]	dependability, security	socio-technical systems (business process)	own metamodel	requirements	semiformal	design and analysis	editor
[37]	any	distributed systems	own metamodel	events	formal	monitoring	none
[26]	any	distributed systems	own metamodel	properties	semi-formal	QoS models exchange	model repository and generic infrastructure to manage NFP
[53]	any	any	-	properties	informal	-	-
[23, 24]	any	service oriented systems	own metamodel	properties (SLA)	semiformal	SLA specification and monitoring	none
[49]	qualitative properties	any	UML profile	NFR as softgoal	informal	softgoal representation and satisfiability evaluation	none
[1]	any	any	own metamodel	events	formal	monitoring	none
[25]	any	any	own metamodel	events	formal	any	none
[29]	performance, security, reliability	service oriented systems	UML profile	properties	semi-formal	analysis, deployment	M2M and M2T transformer

2.3 DEPER Enabler refinements

In the fourth year, the activity on the dependability and performance analysis support (DEPER Enabler) mainly concentrated on enhancing the integration of DEPER in the CONNECT architecture. We investigated deeply the interaction between DEPER and the Synthesis Enabler to close the loop in allowing the dynamic adaptation of the CONNECTor to satisfy dependability and performance properties. Essentially, we exploited the indications provided by DEPER to the Synthesis Enabler on the most suitable dependability mechanism able to overcome deficiencies of the synthesized CONNECTor.

Another form of adaptation involving DEPER consists in enhancing the analysis performed at predeployment time by taking advantage of information available at run-time on real executions of the CONNECTed system. This is the activity performed by the Updater module of DEPER. The process, started in previous years and better consolidated in Y4, is based on the interaction with the Monitor, thanks to which the collection of events relevant for the analysis is achieved.

A significant effort was also devoted to refinement of the DEPER Enabler. In particular, we have worked on the definition of criteria to select an appropriate dependability mechanism to enhance the CONNECTor, in case it fails to satisfy a required non-functional property. In addition, the implementation of the DEPER prototype has been extended so to accept in input the specification of a CONNECTed system in the Colored Automata formalism, currently adopted by the CONNECT Enablers for integration in the overall CONNECT architecture.

2.3.1 Enhancement and adaptation of CONNECTor synthesis

A synthesized CONNECTor, if it exists, provides by construction a correct solution to functional interoperability among the NSs. This is however not sufficient: effective interoperability also requires that such on-the-fly CONNECTed systems provide the required non-functional properties and continue to do so even in presence of evolution. The CONNECTors are not a priori guaranteed to provide the desired non-functional properties for the CONNECTed system. Thus a suitable and adaptive assessment framework is required, and this is provided by the DEPER Enabler. During the fourth year of the project, we have addressed the integration of DEPER and Synthesis Enablers more deeply and have defined the approach to achieve the goal of exploiting the DEPER support to enhance and adapt the synthesis of the CONNECTor to preserve its adequacy with respect to non-functional requirements along time.

A first activity focused on the setting of the overall approach, involving three CONNECT enablers: the Synthesis enabler, which derives on-the-fly a mediator enabling the functional interoperation among heterogenous NSs; the DEPER enabler, which applies stochastic model-based analysis for assessing the desired non-functional properties; and the Monitor enabler, which observes the run-time CONNECTor behaviour. We have discussed on a case study their integrated usage to allow for adaptive analysis accounting for possible inaccurate information or potential evolution of the involved NSs. A cycle involving the three enablers along time has been identified, which starts with automated mediator synthesis, supported by pre-deployment assessment to take appropriate design decisions by providing a priori feedback about how the system is expected to operate. However, the unavoidable high chance of inaccurate/unknown model parameters (since for the NSs only declarative or learned on-the-fly knowledge can be assumed) and possible variations occurring during the system lifetime (due to dynamically assembled components to satisfy an emergent user goal) might result in inadequate analysis results. Therefore, an approach which tries to combine the benefits of both pre-deployment and processing of data obtained at execution time through a lightweight flexible monitoring infrastructure has been developed, to provide feedbacks to the on-the-fly CONNECT system into a continuous loop and thus to support run-time adaptation of CONNECTor synthesis. This activity is documented in paper P3 in Appendix, jointly produced with partner UNIVAQ.

Further progresses have been subsequently made on investigation of a CONNECTor adaptation process which exploits dependability mechanisms. To this aim, a set of dependability mechanisms were already made available in CONNECT in previous years, in the form of a library of adaptation patterns that DEPER suggests to Synthesis to enhance the CONNECTor and make it compliant with the expected non-functional properties. Should DEPER find that the dependability or performance requirement is not satisfied, it starts (through its ‘’Enhancer” module) a new analysis of the CONNECTor model extended

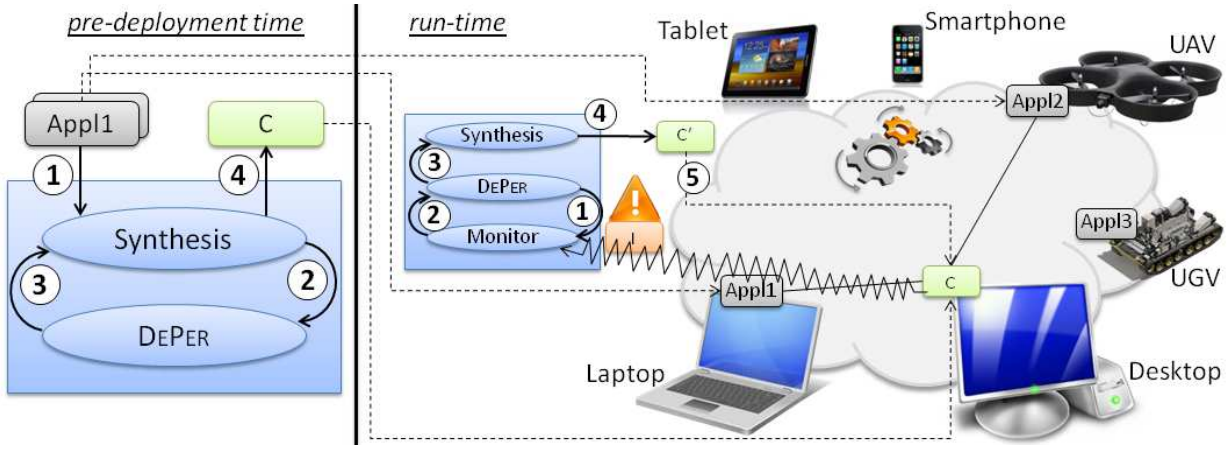


Figure 2.3: Overview of the interactions among DEPER, Synthesis and Monitor Enablers

to include a properly selected dependability mechanism. In case the extended CONNECTor model reveals adequate to satisfy the stated non-functional requirement, the employed dependability mechanism is indicated to Synthesis as a suitable means to enhance the CONNECTor.

We recall that the need for adaptation of a CONNECTor synthesis may arise for the following major reasons: $i)$ the assessment reveals, since the pre-deployment time, that the running infrastructure has deficiencies impairing the requirement (e.g., malfunctions responsible for delays in communications), $ii)$ the data gathered from run-time execution reveal discrepancies with those adopted during the requirement assessment at pre-deployment time, or $iii)$ either the context or the NSs requesting to be CONNECTed have evolved. Once any of these conditions occurs, adaptation is first solved (if possible) at assessment level by DEPER and then reflected by Synthesis on the synthesized CONNECTor.

The loop between Synthesis and DEPER has been finalized and the approach detailed in paper P4 in Appendix, again jointly produced with UNIVAQ. Although no implementation has been performed yet, case studies have been conducted to show the feasibility and the efficacy of the developed CONNECTor adaptation approach.

To provide a graphical view of the interactions and synergic cooperation among the three enablers DEPER, Synthesis and the Monitor GLIMPSE, Figure 2.3 shows such interactions at pre-deployment as well as at run-time. The numbers in the Figure indicate the order in time in which such interactions occur.

Note that, as detailed in deliverable D3.4, the synthesis of the CONNECTor has been also enhanced during this last year to take into account, together with functional concerns, also performance aspects. To this purpose, three strategies have been defined, that can be applied individually or in combination. This process takes place at pre-deployment time, when the need for the generation of a CONNECTor is triggered by heterogeneous NSs requesting interoperability. In view of this additional enhancement, we would like to stress the complementarity of the approaches and their synergic application to reinforce the CONNECTor with respect to the non-functional requirements. In fact, while the strategies put in operation to allow Synthesis to deal with both functional and performance requirements aim at producing a specification of a CONNECTor with the required characteristics, the support provided by DEPER through successive dependability and performance assessments (both at pre-deployment and run-time) aims at coping with problems arising from the execution environment, the uncertainties about the environment itself as known at pre-deployment time and evolution of the working context. These different perspectives make both approaches relevant and strategic.

2.3.2 Adaptive dependability assessment in CONNECT

Pre-deployment assessment is, in general, a crucial activity to drive the system design towards a realization compliant with the required level for quality of service indicators. In fact, it allows for early detection

of design deficiencies, so as to promptly take the appropriate recovery actions, thus significantly saving in money and time with respect to discovering such problems at later stages. Following this rationale, at design time, stochastic model-based analysis is performed by DEPER as a pre-deployment method to support the synthesis of a CONNECTor suitable to allow interoperability among the systems willing to connect under required dependability and performance levels. While the prediction so obtained plays an important role in guiding the building of the CONNECTor, it might suffer from unacceptable inaccuracy because of possibly limited knowledge at analysis time or successive context evolution. This is typically to be expected in dynamic and evolving environment as targeted by CONNECT, hence it would be desirable to have forms of adaptation and refinement of the analysis itself along time. Through monitoring properly selected events at run-time and collecting them along several executions, we can identify changes that require to be accounted for by a new iteration of the model-based analysis.

During this year, we have refined the integration of DEPER with the GLIMPSE enabler, already started in previous years. From GLIMPSE, the ''Updater" module of DEPER receives a continuous flow of data of the parameters under monitoring relative to the different executions of the CONNECTor. The accumulated data are processed through statistical inference techniques. If, for a given parameter, the statistical inference indicates a discrepancy between the on-line observed value and the one used in the model processed at pre-deployment, a new analysis is triggered by instructing the ''Builder" module to update the CONNECTed system model. This is actually another form of adaptation involving DEPER, through which adaptive dependability assessment in dynamic and evolving connected systems is performed. The paper P2 in Appendix describes our approach.

2.3.3 Refinements of the DEPER architecture

The refinements of the DEPER Enabler covered both its definition and the implementation.

During the third year, a set of models of basic dependability mechanisms were devised and exercised in the analysis, in order to enhance the dependability and performance behaviour of the CONNECTor in presence of malfunctions that could arise during the CONNECTed system's execution. In this last year, we have investigated the definition of generic methods for selecting a dependability mechanism suitable to enhance the CONNECTor (among those available), and for identifying elements in the CONNECTor where the mechanism results in higher benefits. Figure 2.4 illustrates the process that is triggered inside DEPER when it receives a request from Synthesis enabler to enhance the CONNECTor.

Concerning the first selection method, an ontology-based approach is indicated, which starts with eliciting an ontology-based characterization of the dependability mechanisms with respect to the three main aspects that have impact on the selection, that is: application constraints, measure under analysis, and fault/failure assumptions. When a mechanism needs to be applied, operate an ontology-based matching to derive the subset of mechanisms that have at least one correspondence in terms of these three characterizing aspects.

Concerning the approach to select where to apply the dependability mechanism, a solution has been devised that stochastically determines the weakest element(s) in the model of the CONNECTor, depending on whether the metric under analysis is a dependability or performance related one. Although not yet implemented in the prototype, these selection methods are shown in the context of a GMES case study. More details about these investigations are in paper P5 in Appendix to this deliverable.

In addition, the implementation of the DEPER prototype has been extended to accept in input the specification of a CONNECTed system in colored automata formalism, as currently adopted by the Synthesis Enabler. So, currently specification of the CONNECTed system to be analysed are accepted by DEPER both in Labelled Transition Systems (LTS, the formalism originally chosen in CONNECT to formalize the abstract CONNECTor and the NSs) and in Colored Automata (currently adopted by the CONNECT Enablers for integration in the overall CONNECT architecture).

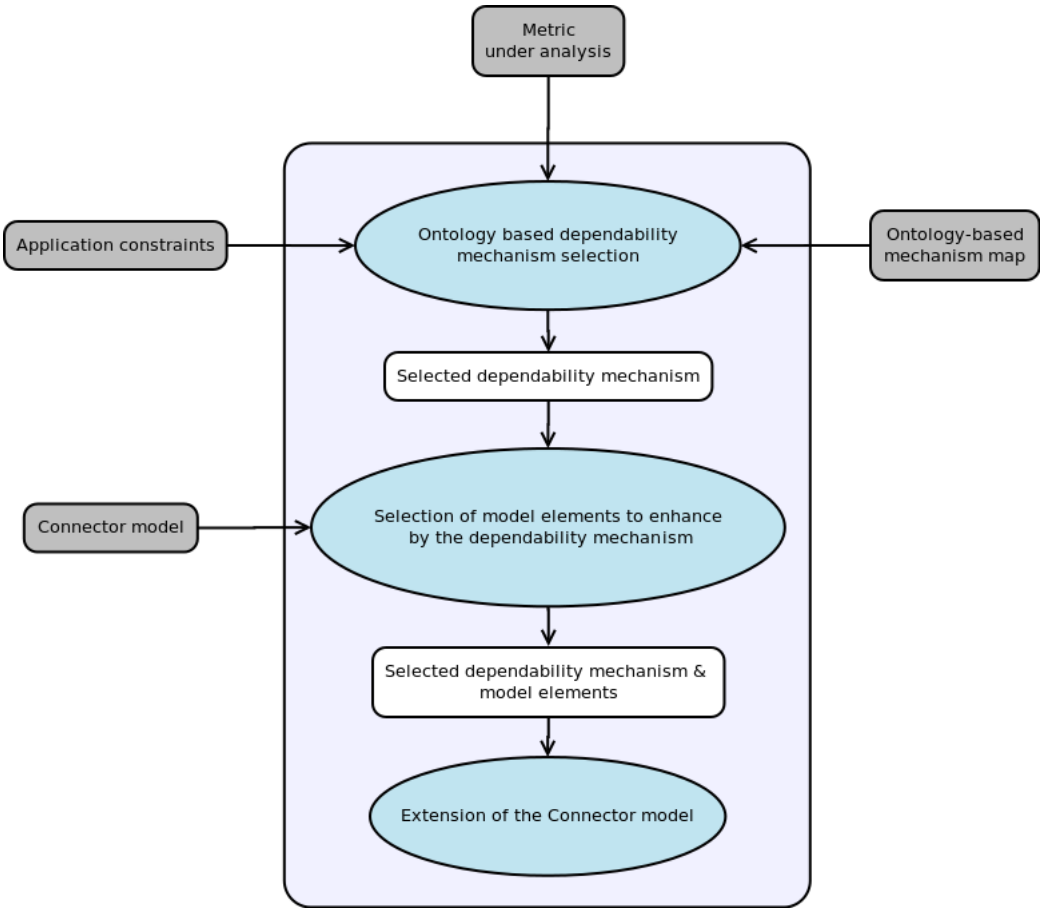


Figure 2.4: Diagram showing the steps performed by DEPER to deal with a request to enhance a CONNECTor

2.4 Quantitative run-time verification refinements

This section describes the contribution towards enabling quantitative verification at run-time, integrated within a system that monitors dependability of CONNECTors. Quantitative verification has an important role to play in this context, because it can guarantee that non-functional properties are satisfied.

In [14], we argued that self-adaptive systems, of which CONNECTed systems are an example, need quantitative verification at run-time. Offline quantitative verification must be complemented by continual online re-verification of self-adaptation decisions at run-time. Two key challenges are that models must be available at run-time, and that efficiency of verification tasks has to be ensured. In this deliverable, we have focused on quantitative models given as Markov decision processes (MDPs) and continuous time Markov chains (CTMCs) and the corresponding run-time verification algorithms.

In Sec 2.4.1 we describe progress towards achieving fast quantitative verification at run-time for self-adaptive MDPs using a symbolic implementation and policy iteration. We have also tackled the problem of incremental model construction. The models are self-adaptive in the sense that they can modify values of parameters that determine the model structures, as well as probabilities, at run-time.

In Sec 2.4.2 we demonstrate, based on a CONNECTor model from a typical CONNECT scenario, how to perform efficient CONNECTor repair at run-time to ensure that a given dependability property (latency) is satisfied. We develop and apply methods for parameter synthesis based on sampling, and evaluate their performance in a CONNECT scenario.

2.4.1 Incremental run-time verification

In [36], we proposed an effective heuristic to improve the efficiency of Strongly Connected Component (SCC)-based value iteration. In general, implementations of value iteration can be improved using Gauss-Seidel schemes, which re-use the latest values for each state that is available during each iteration. These can therefore be used when computing the values for the states within a single SCC. Gauss-Seidel schemes are, by their nature, sensitive to the order in which states are updated during value iteration. In [35], the order taken is arbitrary: for convenience, the implementation simply uses the order in which states had been created during model construction. However, this may not reflect the way that states are connected. Hence, we use the order in which states were visited during the Tarjan SCC detection algorithm. In the experiments shown in [36], this gave the greatest improvement in the speed of convergence of value iteration.

The above incremental verification techniques focused on systems where only transition probability values can be changed at run-time. Now we extended it to deal with system evolution/adaptation that leads to changes in the system structure. We first developed incremental methods for constructing models from high-level system descriptions at run-time and then designed an SCC-based incremental run-time policy iteration to speed up numerical computation when the system structure is changed. We give a summary of this technique in the rest of this section. The detail can be seen in [28].

Incremental model construction

Model construction generates the state space of a model by an exhaustive exploration of its high-level model description. In many cases, model construction plays a significant role in the overall performance of verification, so it is important to consider techniques for improving model construction time. In this section, we focus on models described by a guarded command language. The costly part of model construction is the evaluation of commands and subsequent creation of new states in the MDP being built. Our incremental model construction is designed to operate after relatively small run-time changes to the structure of the MDP. At the level of the high-level model description, we assume that these changes are made by altering *parameters* of the model. These are constants from the model description whose value is not determined until run-time. We only consider changes in parameters that occur in guards of commands, which is a common scenario in practice. For simplicity, we do not consider parameters that affect transition probabilities values. Such changes could be handled using the techniques described in Deliverable D5.2 and D5.3.

Suppose we have a model $\mathcal{M}_1$ obtained for a valuation v_1 of parameters in memory. Our incremental model construction builds a new model $\mathcal{M}_2$ for a new valuation v_2 from $\mathcal{M}_1$ and v_1. We first obtain all

guards G that contain parameters, and search for states in $\mathcal{M}_1$ where a guard $g \in G$ is satisfied by v_1 but no longer satisfied by v_2, and states where g is satisfied by both v_1 and v_2. In the second step, we forward to those states the non-incremental model construction algorithm to compute the state space of $\mathcal{M}_2$.

Remark 1 *The most costly part of this algorithm is the search for all states affected by the change from v_1 to v_2. In the worst case, it has to traverse the whole state space. In practice, case studies often have only a small number of variables dependent of parameters and, by keeping state space ordered by a given variable and using binary search, we can significantly speed up the search process.*

Incremental policy iteration

Now we propose an SCC-based incremental policy iteration for verifying $\mathcal{M}_2$ based on verification results for $\mathcal{M}_1$. As in SCC-based value iteration in D5.2, we first decompose the state space of M_2 into SCCs. Then, we use policy iteration, instead of value iteration, to compute the probabilities for each SCC. As discussed in D5.2 and D5.3, independent SCCs can be processed in parallel to utilise advantage of multi-cores architecture in modern CPUs.

Although policy iteration can use arbitrary memoryless deterministic adversary as a starting point, a good starting adversary can reduce the number of iterations performed before policy iteration terminates. However, it is hard to predict an optimal adversary. To speed up policy iteration for $\mathcal{M}_2$, we use the results from verification of $\mathcal{M}_1$ to guide the selection of the starting adversary. Intuitively, for the set of states that are not affected by the changes from v_1 to v_2, we use the optimal policy obtained in $\mathcal{M}_1$. The reason for doing this is that the behaviour of $\mathcal{M}_2$ in those states might not be affected by the changes. For other states, we use the same strategy as that in normal policy iteration.

2.4.2 Run-time repair by efficient parameter synthesis

Incremental verification techniques can speed up verification at run time. However, an important question relevant to verification remains unaddressed. That is, what can we do if some non-functional properties are not satisfied? The (conceptual) Repairer module in the stochastic model checking engine described in Deliverable D5.3 proposed a means to proceed when the synthesised CONNECTor does not satisfy a given non-functional property. This module focuses on the situation where certain parameters in the CONNECTor can be adjusted. It attempts to compute new values for these parameters such that the CONNECTor synthesised using the new values ensures that the non-functional properties satisfied. In order to effectively determine new parameter values for a new CONNECTor, we assume each adjustable parameter is defined in a bounded domain. To enable usage at run-time, we aim to achieve fast performance of repair. We propose three efficient approaches to solve the parameter synthesis problem with respect to quantitative reachability properties in parametric probabilistic systems. We employ techniques from Monte Carlo sampling and evolutionary computation to obtain inexact, randomised, algorithms that perform well in practice. In particular, we show how to apply two sampling based approaches, i.e., Markov chain Monte Carlo (MCMC [31]) and Cross Entropy (CE) [44], and a swarm-intelligence based method, i.e., the particle swarm (PS) optimisation [33, 48, 51], to the parameter synthesis problem. We evaluate the techniques on CONNECTors from relevant scenarios.

In the rest of this section, we focus on parametric Markov decision processes (PMDPs), which allow transition probabilities as expressions over a set of parameters, rather than concrete values. However, our approach is not limited to PMDP only. We will demonstrate its application over a parametric Continuous Time Markov Chain (PCTMC) model in a CONNECT scenario.

Remark 2 *In theory, there are at least two obvious, "exact" approaches, which would find all good parameter values. (1) Reduce to a mathematical programming problem. (2) Encode the problem in first-order theory of real closed fields, which admits quantifier elimination. However, although theoretically appealing, the scalability of these approaches is rather poor. (They usually only work for up to 10 states, while the problem we are handling is often of magnitude of at least 100,000.) Indeed, the exact complexity of the parameter synthesis problem for PMDPs is an intriguing topic. It is in* PSPACE*, but as one can also easily reduce from the SQUARE-ROOT-SUM or PosSLP problem (which have been open for over 20 years, and*

even their membership in NP *is not known; cf. [3]), it is highly unlikely that a polynomial algorithm exists. Hence, we focus on "inexact" solutions that involve randomised search through the parameter space, yielding some good parameter values efficiently, rather than finding all such values.*

Case study

We have implemented all three algorithms in PRISM. The experiments were carried out on a 64- bit PC with an Intel Core i7-2600 CPU 3.40GHz and 8GB RAM. Our case study is based on the Weather Service scenario presented in Deliverable D6.3 [22]. This scenario is composed of a weather service server that provides weather information, and a number of clients that query the information from the server. As the server and the clients use different protocols, a CONNECTor performing the protocol mapping is needed for each query. Similar to D5.2 and D5.3, we use CTMC to model this scenario. The request operation in the server has a rate λ of the exponential distribution, and the response has a rate μ. The request and response in the client has the same rate λ'.

The weather service server supports 4 request operations:

- The *login* operation takes two parameters (String username, String password) and return an authentication token.
- The *getTemperature* takes one parameter (String token) and returns a string representing temperature, e.g. "20C".
- The *getHumidity* takes one parameter (String token) and returns a string representing humidity, e.g. "55%".
- The *logout* operation take a parameter (String token) and log out the client.

The client has 3 operations:

- The *logToStation* operation takes two parameters (String username, String password) and return an authentication token.
- The *getWeatherInfo* takes one parameter (String token) and returns a pair of strings (one for temperature and the other for humidy).
- The *quitStation* takes one parameter (String token) and log out the client.

As the server can face various hardware and software failures, we added one failure to the probabilistic model and assume that the server can recover from the failure. In this model, the server does an error checking with probability r after receiving a request. Thus the rate for a response operation is changed to $(1-r) \times \mu$. The rate for error checking is $r \times \mu$. We assume that an error happens at rate γ, and the sever is recovered to the error checking state at rate $r \times \mu$. In an error checking state, the server sends a response at rate σ. The CTMC model for the sever, the client and the CONNECTor is shown in Figure 2.5, 2.6 and 2.7 respectively.

Experimental results

In this case study, we fix λ' and μ, and let λ, r, γ and σ be parameters. We also assume that there are n clients sending requests simultaneously. We apply the three methods to search for a good sample, i.e., a set of good parameter values, to satisfy the given average latency for all clients receiving responses to their requests. This property can be specified by a CSL reward formula $R_{\leq latency} \leq l\ [F\ \phi]$, where l is the given latency (threshold) and ϕ is a state where all clients reach their final location "c6" in Figure 2.6. The details about CSL formulae and modelling latency by a reward structure have been reported in Deliverable D5.1 [17] and D5.2 [19].

The experimental results are shown in Table 2.2. In this case study, we investigate how the reward threshold affects the average running time as well as the total number of samples explored. We also vary the model size by adjusting the number of clients, and pick two reward thresholds for each model. For each model and threshold, we run each method five times. We do not take the average as the difference between different runs may be quite large. It is possible that a good sample is found in some run in a

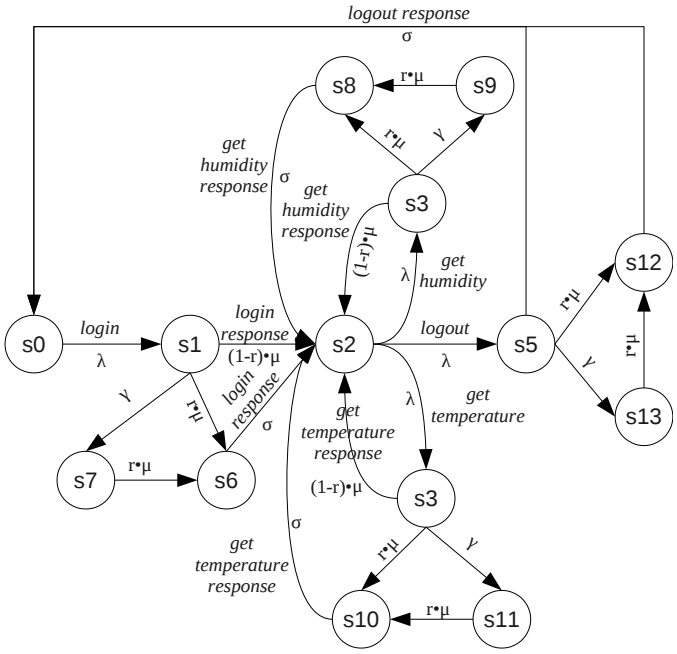


Figure 2.5: The CTMC model for the server

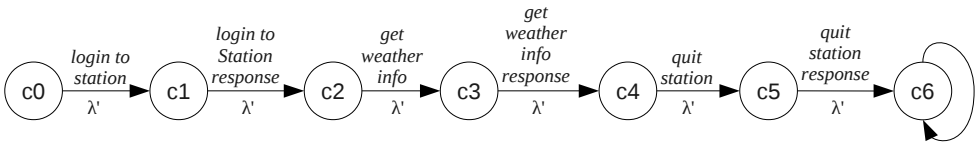


Figure 2.6: The CTMC model for the client

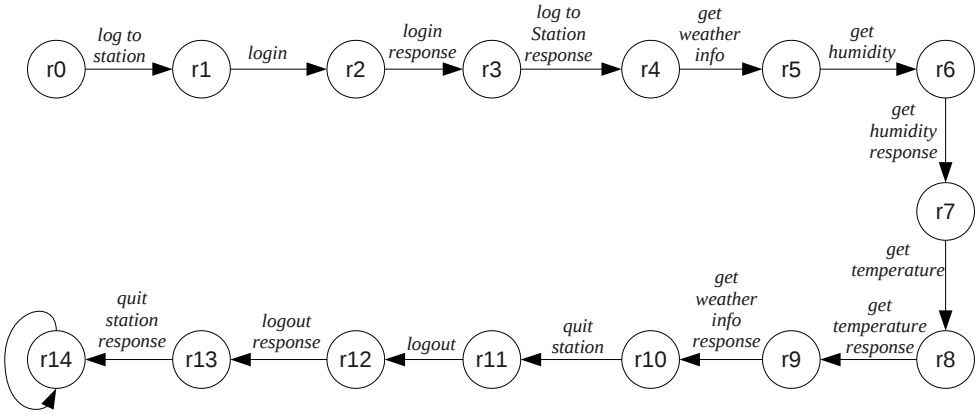


Figure 2.7: The CTMC model for the CONNECTor

short time, but no good samples are found in another run, due to the random nature of our algorithms. As a result, we list all the results in Table 2.2 to give a fairer overview. The size of the CTMC $\mathcal{M}$, denoted $|\mathcal{M}|$, is measured by the number of states of $\mathcal{M}$. "time" and "#samples" are the time (in seconds) spent and number of samples checked before a good sample is found. In the MCMC experiment, we set the maximum number of samples to 2000, namely, the algorithm terminates automatically after 2000 samples are explored; while for the CE method we partition each parameter's domain into four intervals and pick

two samples in each cell. Further, we only allow four iterations. Hence, the CE method terminates when roughly 2000 samples are explored. If no good samples are found when a method terminates, then the corresponding numbers are marked with bold fonts.

Table 2.2: Weather service results – time & $\#$samples vs. rewards threshold

Weather service ($\#$paras$=4$)													
$\#clients$	$\|\mathcal{M}\|$	l	method	round 1		round 2		round 3		round 4		round 5	
				time (s)	$\#$samples	time (s)	$\#$samples	time (s)	$\#$samples	time (s)	$\#$samples	time (s)	$\#$samples
1	23	$R\leq 25.8$	MCMC	**0.518**	**2000**	0.287	572	**0.495**	**2000**	0.335	745	**0.543**	**2000**
			CE	**0.503**	**1836**	**0.54**	**1829**	**0.487**	**1825**	**0.461**	**1837**	**0.467**	**1824**
			PS	0.049	172	0.059	160	0.05	152	0.05	147	0.054	203
		$R\leq 26.8$	MCMC	0.17	207	0.1	23	0.222	346	0.187	263	0.171	203
			CE	0.528	1625	0.319	582	0.307	528	0.107	70	0.32	530
			PS	0.059	165	0.06	169	0.053	129	0.061	111	0.02	11
2	529	$R\leq 29.5$	MCMC	**1.194**	**2000**	**1.129**	**2000**	**1.136**	**2000**	**1.152**	**2000**	1.124	1975
			CE	**1.277**	**1833**	**1.288**	**1829**	**1.268**	**1832**	**1.275**	**1830**	**1.276**	**1830**
			PS	0.154	286	0.128	230	0.129	234	0.093	139	0.215	441
		$R\leq 30.5$	MCMC	0.106	44	0.203	169	0.282	290	0.234	218	0.569	779
			CE	0.575	581	0.612	639	0.611	615	0.969	1097	0.591	598
			PS	0.101	169	0.091	146	0.105	179	0.093	153	0.093	137
3	12167	$R\leq 31.5$	MCMC	**29.339**	**2000**	**28.913**	**2000**	**29.224**	**2000**	**29.467**	**2000**	**29.488**	**2000**
			CE	**34.782**	**1832**	**34.96**	**1828**	**34.921**	**1832**	**34.894**	**1833**	**34.936**	**1836**
			PS	2.113	141	3.569	240	4.086	278	4.002	272	3.399	230
		$R\leq 32.5$	MCMC	23.311	1588	6.486	426	2.974	176	9.93	661	1.796	109
			CE	**34.895**	**1828**	13.249	587	13.113	580	**34.988**	**1830**	**35.128**	**1832**
			PS	3.41	231	2.778	185	2.117	141	2.265	151	3.448	233
4	279841	$R\leq 33$	MCMC	**1028.822**	**2000**	**1030.77**	**2000**	**1046.088**	**2000**	**1024.239**	**2000**	**1027.954**	**2000**
			CE	**1249.162**	**1835**	**1241.261**	**1836**	**1240.472**	**1832**	**1244.047**	**1829**	**1256.268**	**1833**
			PS	115.91	223	60.884	118	116.22	227	226.518	433	170.718	328
		$R\leq 34$	MCMC	375.58	716	579.279	1117	220.751	424	341.109	657	81.194	154
			CE	484.189	624	459.773	578	1025.209	1398	**1263.933**	**1836**	463.601	585
			PS	69.43	134	71.578	139	100.411	187	128.868	244	153.478	296

Let us first examine how the threshold affects the results. It is reasonable that, the looser the threshold is, the larger the number of good samples. Hence, with the high threshold, the time and number of samples are considerably better than with the low threshold. For MCMC and CE, there are many cases where no good samples are found for the low threshold. For the CE method, we also observe that the performance is sometimes quite random (e.g., when $\#client = 1$, $l = 26.8$, the number of samples varies from 70 to 1625).

We then compare the three methods. Overall, the PS method performed the best and was the most stable one. Occasionally, MCMC might beat PS. The MCMC method was quite dependent on the models and thresholds. If the "good region" in the sample space was small, then it was likely that it failed to find one, or took longer time. This is because the MCMC method always started in the centre of the sample space and "walked" towards the good region. The next sample was found based on the current sample. This is different, however, in the PS method, where the next sample was determined by the current swarm. This difference justified the better performance of PS. As it explored the sample space evenly (in each cell), the CE method had to spend quite some time in each cell, even if the cell was not good at all. This increased the overhead and made the CE method not so efficient. However, we comment that, if the good region was scattered over the sample space (which might not be the case in this case study), then the CE method might have better performance.

To summarise, the experimental results suggested that the PS method has stable performance and always managed to find a good sample. It could, most of the time, move very quickly towards the good region, no matter where the initial particle was and where the good region is in the sample space. The CE method usually perform well when the initial cell is near the good region, since it would not waste time in the cells that are not good. In most of cases, CE could find a good sample by narrowing down to the good cell(s). The MCMC method always start from the centre of the sample space. It could quickly find a good sample if the good region is near the centre. In practice, therefore, we should select a method based on the system characteristic: we choose MCMC if we know a good region can be found near the centre of the sample space; choose CE if a good region is close to the initial cell; or choose PS in other cases.

2.5 Security Enabler refinements

In the third year we released a first implementation of the Security Enabler, which can instrument the concrete CONNECTor in such a way that the communications between two NSs always respect the individual NS policies. In the following we summarise the main refinements to the Security Enabler in the fourth year.

2.5.1 Instrumentation

The security instrumentation roughly works by intercepting every message that could potentially violate the security policy, in which case the message is not sent and the CONNECTor moves to the next state. An interesting aspect of the security instrumentation is to make sure that even when the security policy is violated, the CONNECTor maintains a consistent state. This aspect was required by the GMES example: if the drone policy were to be violated while the drone is airborne, stopping brutally the CONNECTor was clearly not an acceptable solution.

A challenge that was revealed during the refinement of the security mechanism is that in practice, the security instrumentation needs to "skip" a state when the policy is violated. Indeed, roughly speaking, each time the CONNECTor is sending a message to an NS, it expects a response back from the NS. However, if the security mechanism intercepts the dangerous message, it is not sent to the NS, which in turn cannot send a response. Hence, after intercepting a dangerous, the security mechanism must ensure that the CONNECTor is in the same state as if it would have sent the message and receive the response.

For instance, in the drone example, the state that sends the message `moveforward` to the drone is the state `As21`:

```
<state>
 <label>As21</label>
 <transition>
  <action>moveforward</action>
  <operation>send</operation>
  <Outputs>
   <Output>
    <simpleType>
     <name>accesstoken</name>
     <type>java.lang.string</type>
    </simpleType>
   </Output>
   <Output>
    <simpleType>
     <name>distance</name>
     <type>java.lang.double</type>
    </simpleType>
   </Output>
  </Outputs>
  <toState>As22</toState>
 </transition>
</state>
```

Once the message is sent, the automaton goes to the state `As22`, which receives the response from the drone, and then goes to the state `As01`, from which other messages can be sent.

```
<state>
 <label>As22</label>
 <transition>
  <action>moveforwardResponse</action>
  <operation>recv</operation><Inputs></Inputs>
  <toState>As01</toState>
```

```
 </transition>
</state>
```

The instrumentation works by traversing the automaton, and checks wether each message is concerned with the security policy. In the above example, the policy has a rule for the message `moveforward`, and therefore the CONNECTor is instrumented in two steps. Firstly, the following guard is added between the tags `<action>` and `<operation>`:

```
<guard><policy>policy_location.xml</policy><exceptionstate>As21_S</exceptionstate></guard>
```

Secondly, the security state `As21_S` is created:

```
<state><label>As21_S</label>
 <transition>
 <action>moveforwardResponse</action><operation>noaction</operation>
 <toState>As01</toState>
</transition></state>
```

Intuitively, in case the policy is violated, the CONNECTor goes from `As21` to `As21_s`, from which it goes directly to `As01` by simulating the response from the drone. The instrumentation proceeds in the same way for all states, creating a new security state whenever required.

2.5.2 Global Variables with the Security Enabler

An important feature of the security mechanism is that it should be as transparent as possible, and in particular should interfere as little as possible with the normal behaviour of the CONNECTor. This feature is required by the nature of the CONNECT itself: since the Security Enabler has on purpose a limited vision of the global architecture, and therefore the security mechanism cannot "take over" the CONNECTor.

For instance, in the Drone example, the security mechanism embedded in the CONNECTor cannot contact directly the drone in order to get its precise location, because such a message could be outside of the expected behaviour, or could even interfere with the current interaction. To illustrate this point, consider a case where the expected behaviour forbids to call twice in a row the method to get the current location of the drone: if the security mechanism makes such a call, that would prevent the client to make a similar call right after, even though the latter call would be perfectly in the expected behaviour.

Hence, we consider that any information required by the security mechanism that cannot be known only by local variable is obtained through *global variables*, which are variables automatically refreshed by the Security Enabler. For instance, in the drone example, the current coordinates of the drone are declared as global variables in the security policy, where `(current_x, current_y)` represents the coordinates of the drone on the map, and where `current_angle` represents the current angle of the drone with respect to the axis North-South. All these variables are initialised to 0.

```
<var>
  <type>double</type>
  <varname>current_x</varname>
  <scope>global</scope>
  <default><dconst>0</dconst></default>
</var>
<var>
  <type>double</type>
  <varname>current_y</varname>
  <scope>global</scope>
  <default><dconst>0</dconst></default>
</var>
<var>
  <type>double</type>
  <varname>current_angle</varname>
  <scope>global</scope>
  <default><dconst>0</dconst></default>
</var>
```

By declaring these variables as global, the security mechanism on the CONNECTor automatically establishes a connection with the Security Enabler, which keeps the value of these variables in a database. The security mechanism keeps a cached value of each variable, and contacts the Security Enabler at a regular interval in order to refresh them. Note that this connection is asynchronous, in order not to block the verification of the policy at run-time. In other words, the security mechanism provides on the one hand the cached value in order to check the security policy, and on the other launches a thread that refreshes these values.

The choice of asynchronicity provides the advantage of evaluating the security policy in an efficient and robust way: even if the Security Enabler would not be available at some point in time, the security mechanism can work with local values without being blocked. However, in order to provide accurate security decisions, the refresh of the variables must be frequent, otherwise the evaluation of the policy could be based on outdated values. This increase in communication might affect the global performances of the CONNECTor, and therefore we monitor the communication rate, in order to adapt it dynamically with the context. This approach is described in the next section.

2.5.3 Interaction with the Monitoring Enabler

As said in the previous section, the security architecture might require frequent interaction between the security mechanism embedded onto the CONNECTor and the security enabler, in order to refresh the cached values of global variables. Clearly, this interaction might have an impact on the performances and the dependability requirement of the CONNECTor, and therefore there must be a tradeoff between the accuracy of the security values and the network performances of the CONNECTor. Hence, the interaction between the CONNECTor and the security enabler is monitored by the monitoring enabler, which can in return advise the security enabler to lower the refresh frequency for the sake of performance. More precisely, the workflow is the following one:

- the security enabler stores in a local database the value of each global variable together with its refresh rate;
- the security mechanism is initialised with a default value for each global variable and a default refresh rate;
- for each variable, a thread is launched, which contacts the security enabler, updates the value of the variable and its refresh rate, and then sleeps for the duration of the refresh rate;
- the security enabler listens to alerts launched by the monitoring enabler, and increases/decreases accordingly the refresh rates of the global variables.

Note that the refresh rate can be different from one variable to another. For instance, the location of the drone needs to be updated quite frequently, while a global variable indicating whether it is raining or not can be updated less frequently.

3 CONNECTability framework

With CONNECT reaching its end, it is now time to wrap up and make a retrospect on the achievements got in assessing and ensuring the CONNECTability of on-the-fly CONNECTed systems. As we recall in the introduction, WP5 addressed the following questions in the realm of highly dynamic heterogeneous systems:

- How do we specify and evaluate non-functional properties?
- How do we verify / ensure the appropriate Dependability and Performance properties?
- How do we enforce / mediate the appropriate Security and Trust levels?

In this chapter we summarise the overall WP5 contribution to CONNECT project, highlighting the key scientific and technological results that we have reported in detail the previous deliverables and in the preceding chapter.

3.1 How do we specify and evaluate non-functional properties?

Our CONNECTability framework is founded on the proper specification of the non-functional properties to be analysed and guaranteed. The central part of this activity consisted in defining the CONNECT Property Meta-Model, or CPMM that supports a model-driven approach to the specification of non-functional properties. This meta-model defines elements and types to specify prescriptive (required) and descriptive (owned) quantitative and qualitative properties that CONNECT actors may expose or must provide.

In the first two years of the project, we focused on the specification of a comprehensive and flexible property metamodel. The key concepts of this metamodel are: Property, MetricsTemplate, Metrics, EventSet, and Event-Type. We separated the property definition from the application domain and its specific ontology. The ontology is linked to the Property metamodel via the EventType entity that models a generic event type where the terms of the application-domain ontology will be used. We distinguished in CPMM the Metrics and MetricsTemplate concepts. The MetricsTemplate is defined upon a generic set of events/operations, that is not coupled with a particular application domain. This general part of the definition is specialized by the metric that instantiates those general concepts (templateParameters) with application-based events/operations (metricsParameters).

During the third year, we worked into the following two directions:

- several improvements have been embedded into the metamodel to better specify properties and complex events. Specifically, we introduced a complex events language defining operators for combining simple and complex events. We also provided a detailed mapping of these operators with those of two existing complex events languages that are GEM and Drools Fusion;
- to fill the gap between the definition of properties of interest and their concrete usage within the CONNECT architecture, we provided automated Model2Code transformations for translating CONNECT properties into lower-level monitor configuration directives.

In the last year we refined and completed the Model2Code transformations. In addition, we concentrated on more deeply evaluating the comprehensiveness and flexibility of CPMM with respect to similar approaches. To this aim we performed a systematic survey of the literature on property metamodels and complex events languages. We deduced that existing metamodels deal with only a subset of the properties addressed in CPMM or do not support transformational approaches.

The property models conforming to CPMM and expressed in a machine-processable language have been used as the exchange language among CONNECT Enablers to communicate and manage non-functional properties. Specifically, properties defined according to CPMM have been used:

- as input for the dependability Enabler to verify specified CONNECTability properties;
- as instrumentation for the monitoring Enabler that generates suitable probes to monitor useful properties on the CONNECTors;

- as input for the synthesis Enabler to guide the synthesis of a new CONNECTor or the selection process in case suitable CONNECTors are already available.

Summarizing, the main results of this work include:

- a **Property Meta Model** that allows for the **specification of CONNECT properties spanning over dependability, performance, security and trust**. We make available CPMM eCore model equipped with a dedicated editor (as an Eclipse Plugin), for using existing models and deriving new property and metrics from CPMM;
- a machine processable specification language that allows for **defining complex events models involved into non-functional properties**. The proposed language is specified as part of CPMM, but it can be used in isolation to specify events that are not necessarily tied to a property modeling. This specification language combines features of two existing event specification languages that are GEM and Drools Fusion and in addition presents new features not included in the considered languages. In particular, we have defined operators that allow for modeling a temporal relationship (as those of Drools Fusion) and operators that allow for combining simple or complex events (as those of GEM), in addition we have identified situations of interest not covered by the operators of GEM and Drools Fusion that have been formalized through new operators;
- **automated procedures (in form of Model2Code transformations)** that translate models conforming to CPMM (or part of them) into Drools rules, which are used to configure the GLIMPSE monitoring infrastructure for run-time verification of CONNECT properties. The aim of such transformation is to allow for the dynamic configuration of GLIMPSE whenever a new property to be monitored is specified and introduced in CONNECT. This transformation has been implemented using the Acceleo technology;
- a **systematic literature review** comparing CPMM with existing similar approaches.

3.2 How do we verify / ensure the appropriate Dependability and Performance properties?

According to its widely recognized definition, dependability is ''the ability to deliver service that can justifiably be trusted", thus calling for means to verify and validate the developed system with respect to specified dependability properties. In CONNECT, model-based analysis has been employed to this purpose, an approach applicable since the early stages of system development and therefore able to promptly point out deficiencies of the design with respect to non-functional requirements.

State-based stochastic methods and stochastic model checking have been exploited in CONNECT to enrich the variety of dependability analyses. We have conducted a detailed comparison between these two approaches, based on the Terrorist Alert scenario and the distributed market place scenario. We performed various dependability and performance analysis using PRISM, a stochastic model checker, and Mobius, a state-based stochastic analysis tool. First, both approaches are used to validate two basic dependability and performance properties. Next, additional properties are checked by the appropriate approach, selected according to its ability to handle the specific type of analysis.We concluded from the comparison that the two approaches are complementary in assessing dependability and performance properties.Indeed, the different formalisms and tools implied by the two methods allow: (i) on the one hand, to serve as a basis for a detailed analysis, by focusing on aspects such as the level of abstraction, scalability and accuracy, for which the two approaches may show different capabilities; and (ii) on the other hand, through the inner diversity, provide cross-validation to enhance confidence in the correctness of the analysis itself.

For what concerns dependability and performance analysis, the key result achieved during the project's life has been the **definition and development of the DEPER Enabler, to assist the whole CONNECT environment towards the synthesis and deployment of a Connector suitable to satisfy non-functional requirements.** Also, a prototype of DEPER has been realized and integrated, to a certain extent, in the CONNECT architecture.

DEPER has been designed following a modular approach, to promote efficiency and rigorous development of the different functionalities performed to achieve dependability and performance analysis. Six modules do compose this Enabler: Builder, Analyser, Evaluator, Enhancer (for the state-based stochastic analysis engine), Repairer (for the stochastic model checking analysis engine) and Updater.

The major features of DePer are:

1. automated translation of specifications of the CONNECTor and the Networked Systems involved in the communication in dependability and performance models suitable to be analysed to get quantitative assessment of the metrics of interest. The implemented prototype is based on both the Coloured Automata and Labelled Transition Systems (LTS) formalisms for the specification of the CONNECT entities taken in input, and the stochastic Activity Network (SAN) formalism to express the dependability and performance models, which are then solved through the Mobius tool. The metrics under evaluation are formalised through the CONNECT Property Meta-Model (CPMM);

2. ability to **combine classical pre-deployment analysis with on-line adaptation through recalibration of model parameters on the basis of data relative to real executions along time**. This is achieved through the cooperation with GLIMPSE, the CONNECT lightweight monitor enabler, which is properly instructed by the Updater module of DEPER to observe relevant events at run-time and convey related data on such observed events. The dynamicity and evolution of the CONNECT environment lead to potential sources of uncertainty, which undermine the accuracy of the pre-deployment analysis. It is therefore crucial to be investigate on adaptive dependability assessment, with re-calibration and refinement of the dependability and performance prediction along time, as done within DEPER. The integration of DEPER with GLIMPSE has been implemented at prototype level;

3. ability to **embed selected dependability mechanisms in the CONNECTor model under assessment**, to evaluate their efficacy to improve the synthesised CONNECTor, in case the analysis reveals that dependability or performance metrics are not satisfied. If the extended CONNECTor model reveals adequate to satisfy the stated non-functional requirement, the employed dependability mechanism is indicated to Synthesis as a suitable means to enhance the connector. This support provided by DEPER aims at adapting the connector in response to problems arising from the execution environment, the uncertainties about the environment itself as known at pre-deployment time and evolution of the working context. The Enhancer module of DEPER, implemented in the prototype with a selection of five basic dependability mechanisms, is in charge of this activity. Although the full interaction between DEPER and Synthesis to realize this adaptation process has not been implemented yet, extensive discussion and case studies have been conducted and documented in papers.

Due to the continuous evolution of the context with which a CONNECTor interacts, offline (static) analysis result may be invalidated after a CONNECTor is deployed. The accuracy of the analysis results can also be limited because unpredictable phenomena may affect the system during its operation. Therefore, the analysis typically needs to be refined or repeated with data obtained from real system executions. We introduced the concept of **(online) quantitative run time verification**, so that changes to the evolving system can be taken into account and verified at run time. When running verification tasks at run time, it is important to optimise the execution time, particularly when the system is evolving, in case subsequent changes occur before the task terminates. We presented an approach for incremental verification which **improves the performance of verification at run time by reusing results from previous verification runs** to obtain fast accurate results during the evolution of a CONNECTed system. Our approach decomposes a model into strongly connected components (SCCs) and identifies unaffected SCCs during system evolution, which do not need to be processed during incremental verification. In addition, our approach also improves the efficiency of SCC-based offline probabilistic model checking, and minimises the computation across multiple verifications using the improved technique.

The first implementation of the incremental verification technique used explicit-state data structures to store the state space and transition relation, which imposes a limit on the size of models that can be handled. In order to overcome this limit, we provided a **symbolic implementation of our technique**, based on binary decision diagrams (BDDs) and extensions such as multi-terminal BDDs (MTBDDs). A

particular difficulty here was that the Tarjan algorithm for identifying SCCs is known to be poorly suited to symbolic implementation. For example, some operations cannot be performed efficiently with BDDs, notably association and update of an integer index to a state. Various SCC decomposition algorithms have been proposed, specifically for implementation with BDDs.Unfortunately, they do not explore SCCs in reverse topological order, and it is very slow to generate this order once the SCCs are stored as BDDs. We therefore proposed a novel hybrid Tarjan algorithm, which combines symbolic and explicit-state data structures in order to keep overhead to a minimum for efficiency.

Our incremental verification technique is initially limited to CONNECTed systems that are only subject to **changes in probability values** at run time. We have extended it to deal with system evolution/adaptation that leads to **changes in the system structure**. We first developed incremental methods for constructing models from high-level system descriptions and then designed an SCC-based incremental policy iteration to speed up numerical computation when the system structure is changed. The incremental model construction technique computes all states that have to be visited during incremental verification. The incremental policy iteration technique decomposes the system into SCCs, and is performed incrementally by reusing policies between verification runs.

Incremental verification can speed up verification at run time. However, an important question relevant to verification remains unaddressed. That is, what can we do if some non-functional properties are not satisfied? The (conceptual) Repairer module in the stochastic model checking engine proposed a means to proceed when the synthesised CONNECTor does not satisfy a given non-functional property. This module focuses on the situation where certain parameters in the CONNECTor can be adjusted. It attempts to compute new values for these parameters such that the CONNECTor synthesised using the new values ensures that the non-functional properties satisfied. In order to effectively determine new parameter values for a new CONNECTor, we assume each adjustable parameter is defined in a bounded domain. To enable usage at run-time, we aim to achieve fast performance of repair. We propose three efficient approaches to solve the parameter synthesis problem with respect to quantitative reachability properties in parametric probabilistic systems. We employ techniques from Monte Carlo sampling and evolutionary computation to obtain inexact, randomised, algorithms that perform well in practice. In particular, we show how to apply two sampling based approaches, i.e., Markov chain Monte Carlo and the cross entropy method, and a swarm-intelligence based method, i.e., the particle swarm optimisation, to the parameter synthesis problem. We evaluate the techniques on CONNECTors from relevant scenarios.

3.3 How do we enforce / mediate the appropriate Security and Trust levels?

One major goal in CONNECTability assurance is to guarantee that whenever a CONNECTor is provided, the connection among networked systems works as expected from the security prospective.

Briefly, along the whole project we have:

- analysed the **threat models of the CONNECT framework from the security prospective**;
- extended the Security-by-Contract framework also considering trust relations that can be active among different agents of the framework. As result we have provided the **Security-by-Contract-with-Trust framework**;
- provided a description of a **trust negotiation framework** for establishing trust relations among agents of the connect framework;
- provided a framework for speeding up the synthesis of **secure CONNECTors that are able to also deal with cryptography**;
- provided an implementation of the security enabler.

In the first two years of the project we have mainly focused on the analysis of the threat models of the scenario depicted by the CONNECT framework. Indeed, the CONNECT world is composed by networked systems that ask to establish a communication and a CONNECTor, generated and provided by the CONNECT infrastructure composed by enablers, that allow the networked systems to communicate between

each other. Hence we have three set of entities, i) the networked systems, ii) the CONNECT enablers, and iii) the CONNECTor. Investigating the trust relations among these entities we have depicted several possible scenarios in which security aspects have to be considered and assured.

We work at two different levels within the CONNECT architecture:

- at synthesis level, by providing an automatic strategy for generating security features for the CONNECTor in order to deal with cryptographic primitives;
- at execution level by extending the Security-by-Contract framework by taking into account trust relations. We describe the Security-by-Contract-with-Trust (SxCxT) framework.

During the first three year of the project we mainly work at execution level. Indeed, the S×C×T paradigm is developed for guaranteeing at run-time that:

- local private policies, that are not shared neither with the CONNECT infrastructure, are satisfied. Example of local private policies are private policies regarding, for instance the GPS coordinates of the NS itself, the right for accessing to the private data stored on the device of the NS, and so on.
- none NS tries to attack the other NS by manipulating the part of the CONNECTor running on it. Indeed, let us suppose that one of the two NS is a malicious agent. When it receives the part of CONNECTor that it is in charge to run, the NS manipulates it in order to attack the other NS during the communication.

Several trust models can be used in order to model and establish trust relations among agents of the considered framework. Within the SxCxT approach above outlined, we have introduced a generic trust meta-model as a basis to express and to compose a wide range of trust models, so that heterogeneous trust management systems belonging to different NSs can interoperate transparently. To this aim, novel mediation algorithms have been developed to overcome the heterogeneity between the trust metrics, relations and operations associated with the composed trust models. We proposed an access control policy negotiation mechanism for allowing two NSs to state and enforce their access control policies to their services. These access control policies are enriched with weights for expressing trust measures. In particular, in order to give full semantics to these family of languages, we use a weighted version of Datalog where the rules are enhanced with values taken from a proper c-semiring structure. The proposed framework is used when two NSs need to communicate but do not know each other. The Security enabler can help them by comparing their access control rules in an automatic and formal way in such a way a trust relation among them is established.

Concerning the assurance of desired trust levels, we have introduced **TMDL (Trust Model Description Language) as the basis to express and to compose a wide range of trust management systems and thereby support trust management across heterogeneous networked systems.** Using TMDL, two heterogeneous trust models from two NSs willing to be CONNECTed can be modeled, composed and mediated. The composition is specified in terms of mapping rules between roles of the original models. Rules are then processed by a set of mediation algorithms to overcome the heterogeneity of the trust metrics, relations and operations associated with the composed trust models. We have provided a complete XML representation for TMDL.

We also implemented several dedicated tools that (i) guide developers to **check and create a valid TMDL description**; (ii) automatically generate from such description the Java code of the corresponding trust management system; and (iii) enables the **composition of any given trust management system according to given mapping rules**. As part of such framework, we have also developed a **TMDL editor** that guides developers to create a valid and correct TMDL description which can serve to automatically generate the JAVA code of the corresponding trust management system [45].

3.4 Prototypes

The above summarised scientific advances have all been instantiated in several supporting tools. The software is released as an integral part of this deliverable. In the associated Appendix-Prototypes document D5.4P, we provide the list of released tools and enablers, along with essential information and the respective URLs from which they can be downloaded. Namely they include:

- CPMM eCore & Editor for CONNECT properties;
- Automated support for Model2Code transformation from CPMM to Drools;
- the DePer Analysis prototype, which instantiates the Dependability&Performance architecture;
- PRISM CONNECT Bundle, which is a prototype of the incremental verification technique;
- the SxCxT infrastructure, including the Security Enabler Web service, the Security Lib and the Instrumentation client interface;
- the iMTrust set of tools, including the associated iTMDL Editor.

4 Evaluation

With reference to previous assessment reports (see Deliverable D6.3 [22]), the objectives for dependability (CONNECTability) assurance in CONNECT have been decomposed into four sub-objectives, following the task structure of the WP. The four sub-objectives have been expressed as follows:

1. A successful approach for dependability assurance in CONNECT will consist in having a set of clear qualitative and quantitative metrics to apply to CONNECTed systems such as security and privacy levels.

2. The approach should automate the creation of validation suites for end-to-end monitoring of interactions, which can be measured by experimentation with the WP6 scenarios.

3. The effectiveness of the approach will also depend on the robustness of the verification and validation techniques to such disruptions as system evolution, faults and malicious attacks.

4. The approach should provide trust mechanisms that handle dynamic compositions and security policy languages with sufficient expressivity.

In this chapter, refining over previous statements [22], we make a final assessment of the extent to which the above sub-objectives have been achieved along the CONNECT whole duration.

4.1 Objective 1: Qualitative and quantitative metrics

Assessment criterion:

As stated in the DOW, the assessment will consider clearness, i.e. lack of ambiguity and degree of formalisation, of definition of CONNECT related non-functional properties. This is an important feature to support the automated analyses foreseen within the CONNECT architecture (see e.g., Objective 2 below).

Methodology:

To evaluate the achievement of the above objective, we need to assess the capability and ease of use for prospective CONNECT users to refer to the provided formalism for specification of properties and metrics as an input to their analysis and assessment tools. The methodology for assessment will follow two directions: flexibility and breadth. In particular, the degree of success will be the higher the more flexible the specification is with respect to automated transformation into different formalisms, and the broader the range of properties that can be expressed.

Final assessment:

The CONNECT Property Meta Model (CPMM) permits to express the desired non-functional properties in a clear, formal way. CPMM is now complete and expressive enough for defining dependability, performance and security properties. Trust properties have not been included, as the activity in Task 5.4 has been discontinued in Y4.

Concerning flexibility, we have demonstrated how easily CPMM definitions can be used as an input to the GLIMPSE monitor, by exploiting the automated translator from CPMM into Drools Fusion. Examples of transformations are included in [11] and can also be found on line at the CPMM site [41]. Other transformations have been as easily performed by exploiting the same CPMM modeling as an input for the KLAPERSUITE tool. KLAPERSUITE is based on the KLAPER model [16], a convenient pivot model that fills the gap between user design model and quality analysis model. KLAPERSUITE can provide several kind of analysis and results. We have exploited CPMM models of non-functional properties in the context of another European project in collaboration with a team from Politecnico di Milano. The approach to translate from CPMM to KLAPER is described in [5, 6].

Concerning breadth, the definition of CPMM has been carried out by taking into account related existing conceptual models and unifying, into one comprehensive framework, all aspects considered relevant for CONNECTed systems. To assess the comprehensiveness of CPMM meta-model we have compared it with most relevant competitive notations. The results are summarised in Table 2.1 and detailed in paper P1 in Appendix.

4.2 Objective 2: Automation

Assessment criterion:

Degree of automation in guiding and performing run-time validation of CONNECTed systems through monitoring.

Methodology:

To evaluate the achievement of the above objective, we intend to provide experimental evidence through the CONNECT scenarios, showing demonstration as far as possible of a full story from user-desired non-functional properties down to run-time monitoring of those properties for a CONNECTed system, passing through all the steps of discovery, synthesis, analysis and deployment.

Final assessment:

During the fourth year, evaluation of the degree of automation of run-time validation has progressed along various directions. The approach can be easily adopted for all Enablers in the CONNECT architecture thanks to the adopted publish/subscribe paradigm, and in fact all along the project we could describe or demonstrate the interaction between GLIMPSE and Synthesis [12], GLIMPSE and Deper (see Paper P2 in Appendix), Security (see Section 2.5), Learning [10], as well as a full cycle among synthesis+Deper+monitor (see Paper P3 in Appendix).

As planned, we finally also set up a demonstrator of a full cycle scenario as part of GMES, in which we show that the non-functional requirements collected off-line from the NSs, are first taken in charge by the relevant Enablers (DePer, Security) and then converted into the rules to be monitored as CPMM properties. Within the developed WP6 scenarios, in particular concerning the part relative to controlling the Drone flight, we exemplify two subscenarios: one case in which DePer can validate at run-time latency properties by automatically instructing the monitor, and another case in which the Security Enabler can adjust its own performance following a warning from the monitor.

4.3 Objective 3: Robustness of V&V

Assessment criterion:

How robust are proposed V&V frameworks to handle connection disruptions as system evolution, faults and malicious attacks.

Methodology: To evaluate the achievement of the above objective, we intend to provide evidence in two directions: i) by implementing at the prototype level and demonstrating within the CONNECT scenarios how DEPER can adapt the analysis in presence of system evolution and accidental faults. This has been possible for those features which have reached a more mature level, given the developments carried on during the project lifetime; and ii) demonstrating in the context of CONNECT scenarios those abilities which have been tackled in the last periods, and for which there weren't enough time and resources to transfer them at prototype level.

Final assessment:

Concerning the assessment of the robustness of V&V techniques against disruptions, we have pursued two research directions:

- i) self-adaptation of the analysis to uncertainty and evolution of the involved actors. This is an important ability in the CONNECT context, where the dynamicity and evolution of the targeted environment lead to potential sources of uncertainty, which undermine the accuracy of the pre-deployment analysis. To cope with this issue, adaptive dependability assessment has been investigated, which exploits run-time monitoring to re-calibrate and enhance the dependability and performance prediction along time. In brief, the picture of the synergic usage is the following. At design time, stochastic model-based analysis is performed as a pre-deployment method to support the synthesis of a CONNECTor suitable to allow interoperability among the systems willing to connect under required dependability and performance levels. While the prediction so obtained plays an important role in guiding the building of the CONNECTor, it might suffer from unacceptable inaccuracy because of possibly limited knowledge at analysis time or successive context evolution. Through monitoring properly selected events at run-time and collecting them along several executions, we can identify changes that require to be accounted for by a new iteration of the model-based analysis. In addition

to defining the approach that synergistically joins the analysis through DEPER with the run-time observations performed by GLIMPSE, we have integrated the features of the two enablers at prototype level and exploited them in selected case studies from CONNECT.

- ii) supporting CONNECTor adaptation. Again, due to evolution and (partially) unknown info about the networked systems willing to interoperate, it is important to have a support that guarantees that the satisfaction by the synthesised CONNECTor of the non-functional requirements be preserved along time. During this fourth year, we have concretized the approach that integrates DEPER and Synthesis enablers toward reaching a continuous adaptation process of the CONNECTor based on the dependability and performance analysis feedback. Specifically, a dependability mechanism (among a set of available ones) is suggested by DEPER to Synthesis when a performance or dependability violation notification is received, which is stocastically proved to be adequate to solve the problem. Extending the CONNECTor to include the indicated mechanisms ensures its continued and correct usage in successive executions. Although not transferred at prototype level, we have assessed the efficacy of the approach through CONNECT case studies in the GMES context.

Although it was not possible to reach the same level of maturity from the point of view of their reflection in the prototype (given the constraints on time and resources already allocated), both features offered by DEPER to support analysis in dynamic and evolving context have shown feasible and with relevant potentialities in CONNECT-like contexts.

4.4 Objective 4: Expressing trust and security

Assessment criterion:

- Capability for trust management to support dynamic composition
- Capability for security enabler to handle relevant security policies.

Methodology:
We intend to assess the successful satisfaction of both above requirements by experimental evaluation of both the trust management system and the security enabler on the GMES scenario.

4.4.1 Trust Assessment

The activity has been discontinued in the last year, therefore we have no further evaluation to report. The activity on Distributed Trust Management (Task 5.4) has been concluded in the third year. As reported in D5.3, we have defined a trust meta-model that allows the rigorous specification of trust models as well as their composition. The resulting composite trust models enable heterogeneous trust management systems to interoperate transparently through mediators, as we show in [45].

4.4.2 Security Assessment

From the security perspective, the objective was to be able to handle relevant security policies. This objective has been fulfilled by defining a simple XML-based security policy language, where the elements of the language consider directly the messages exchanged by the CONNECTor at run-time. For instance, we have implemented a security policy limiting the altitude of the drone, and another one limiting the area accessible by the drone.

In addition, the policy can be directly integrated in the description of the NS, and automatically taken into account when synthesising the CONNECTor, by calling the security enabler, which is in charge of the instrumentation. The interaction between the CONNECTor and the security enabler is transparent for the NS, since this interaction is performed asynchronously, and therefore is not blocking. Moreover, the instrumentation of the policy is done within the CONNECTor in order to always keep the CONNECTor in a consistent state. Operations that would violate the security policy are not forwarded, meaning that the security policy is not violated, and the CONNECTor behaves as if the corresponding operation has never been sent.

Finally, it is worth noting that the security mechanism is built in a robust non-blocking way, meaning that the function call checking whether an operation is violated never returns an exception (as they are all caught at a lower level), and returns false by default, meaning that in case of any problem, the CONNECTor works as if everything is secure. We followed this approach in the context of the GMES scenario, since it is dealing with emergency conditions, but we could clearly turn it around, by considering that in case of any problem, an operation should be consider as non-secure, and therefore blocked.

5 Conclusions and Future Work

As this is the final deliverable for WP5, the time has now come for a quick wrap up and hints for future work. We have already made a comprehensive summary of the results achieved in Chapter 3. Here we make a short recap of main concepts and discuss how we intend to continue the research along the various started threads.

The activity has been articulated along parallel and inter-related tracks. The work related to Task 5.1 has given rise to an intense activity for model-driven treatment of non-functional properties. As we could not find some reference conceptual model that could serve for the purpose of CONNECT, we started in Y2 the definition of a meta-model, built on top of the state of art advances. This task has revealed a very fruitful research activity, based on a tight collaboration between ISTI-CNR and Univaq. The resulting meta-model is more comprehensive and complete than any existing competitor model, as we could verify from a recent systematic survey (reported in paper P1 in Appendix). Far from being complete, this activity will continue in several directions: i) so far we have exemplified its usage for synthesis and monitoring; in future we intend to use the meta-model all along the lifecycle of dynamic heterogenous systems, including testing, requirement analysis; ii) we need to investigate the possibility to enhance CPMM to also deal with state-based properties; iii) we will continue experimentation in different scenarios and context: currently we are employing CPMM for driving analysis and monitoring of choreographies.

Also, there have been intense research and development activities carried out in dependability and performance analysis in heterogeneous, open and (partially) unknown networked contexts. During CONNECT lifetime significant outputs have been produced. Among such results, there are: i) the methodology for the refinement of the analysis during run-time, by exploiting the synergic usage of the Monitoring enabler, to cope with possible inaccuracy or evolution of the model parameters; ii) the approach to enhancement and adaptation of the CONNECTor through automatic extension of the CONNECTed system model with models of dependability mechanisms useful to cope with malfunctions and changes in the context and NSs requesting services to CONNECT; iii) the strategies to automate the selection of the proper dependability mechanism in the available set, according to NSs needs. Of course, consolidation of, and advancements to, the proposed solutions would be desirable, since, given the complexity of the problems tackled, the investigations have been conducted incrementally to manage such complexity. Further work is therefore planned, both in terms of addressing higher levels of complexity for some of the proposed solutions in order to better fit real contexts, and in detailing and concretizing aspects currently kept at a rather high level.

Offline quantitative verification must be complemented by continual online re-verification of self-adaptation decisions at run-time. We have introduced several techniques for incremental verification techniques. Future work will include formulating and implementing efficient methods for application steering for distributed probabilistic systems modelled as Markov decision processes with respect to properties such as ensuring minimum level of expected energy consumption. For this purpose, we will extend incremental model construction and policy iteration developed as part of this deliverable. We will also further develop model repair based on parameter synthesis for broader classes of models, with the view to apply it in run-time quantitative verification of adaptive systems and parameter estimation.

The security results obtained during the CONNECT project also open several interesting leads for future work. Firstly, the way the CONNECTor reacts to a security violation could be improved by considering more complex scenarios, where the security mechanism would interact directly with the NSs instead of simply suppressing messages. For instance, in the UAV example, it would not be unreasonable to expect the CONNECTor to send an emergency landing message when the policy is constantly violated. However, such an approach requires to modify the overall behaviour of the CONNECTor, which could lead the automaton to an inconsistent state. Another interesting problem is the distributed enforcement of security policies, when a policy is expressed over several CONNECTors, and therefore requires a communication between the different controllers. This problem is currently addressed using the Security Enabler, which acts as a coordinator, but by considering that two CONNECTors that need to interact are in fact two NSs, we could leverage the CONNECT architecture and synthesize a new CONNECTor that would automatically connect the two CONNECTors. Finally, the problem of security variable refresh is quite challenging, since it requires to establish a trade-off between security accuracy and efficiency. The usage of quantitative techniques to calculate such a trade-off is an emerging approach in security, and a risk model could be used to address this problem.

Bibliography

[1] Drools fusion: Complex event processor. http://www.jboss.org/drools/drools-fusion.html.

[2] Acceleo. http://www.eclipse.org/acceleo/.

[3] E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 38(5):1987–2006, 2009.

[4] D. Ameller, X. Franch, and J. Cabot. Dealing with non-functional requirements in model-driven development. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pages 189 –198, 2010.

[5] C. Bartolini, A. Bertolino, A. Ciancone, G. De Angelis, and R. Mirandola. Non-functional analysis of service choreographies. In *Proc. of PESOS 2012*, Zurich, Switzerland, June 2012. IEEE-CS. – to appear, accepted for publication on 2012, Mar. 20.

[6] C. Bartolini, A. Bertolino, A. Ciancone, G. De Angelis, and R. Mirandola. Quality requirements for service choreographies. In *Proc. WEBIST 2012*, Porto, Portugal, Apr. 2012. SciTePress.

[7] D. Basin, M. Clavel, J. Doser, and M. Egea. A metamodel-based approach for analyzing security-design models. In *Models*, pages 190–204, 2007.

[8] D. Basin, M. Clavel, J. Doser, and M. Egea. Automated analysis of security-design models. *Inf. Softw. Technol.*, 51(5):815–831, May 2009.

[9] A. Bertolino, A. Calabrò, A. D. Marco, F. Lonetti, and A. Sabetta. Meta-Modeling of Non-Functional-Properties. *To be sumbitted to Software and Systems Modeling*.

[10] A. Bertolino, A. Calabrò, M. Merten, and B. Steffen. Never-stop learning: Continuous validation of learned models for evolving systems through monitoring. *ERCIM News*, 2012(88), 2012.

[11] A. Bertolino, A. Di Marco, and F. Lonetti. Complex events specification for properties validation. In *Proceedings of 8th International Conference on the Quality of Information and Communications Technology (QUATIC)*, pages 85–94, 3-6 September 2012, Lisbon, Portugal.

[12] A. Bertolino, P. Inverardi, V. Issarny, A. Sabetta, and R. Spalazzese. On-the-fly interoperability through automated mediator synthesis and monitoring. In T. Margaria and B. Steffen, editors, *Proc. ISOLA 2010 - Leveraging Applications of Formal Methods, Verification, and Validation*, volume 6416 of *LNCS*, pages 251–262. Springer, 2010.

[13] G. Beydoun, G. Low, H. Mouratidis, and B. Henderson-Sellers. A security-aware metamodel for multi-agent systems (mas). *Information and Software Technology*, 51(5):832 – 845, 2009.

[14] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Communications of the ACM*, 55(9):69–77, September 2012.

[15] S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Data & Knowledge Engineering*, 14(1):1–26, 1994.

[16] A. Ciancone, A. Filieri, M. L. Drago, R. Mirandola, and V. Grassi. Klapersuite: An integrated model-driven environment for reliability and performance analysis of component-based systems. In *TOOLS (49)*, volume 6705 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2011.

[17] CONNECT Consortium. Deliverable 5.1 – conceptual models for assessment & assurance of dependability, security and privacy in the eternal CONNECTed world, 2010.

[18] CONNECT Consortium. Deliverable 4.2 – Further development of learning techniques, 2011.

[19] CONNECT Consortium. Deliverable 5.2 – Design of Approaches for Dependability and Initial Prototypes, 2011.

[20] CONNECT Consortium. Deliverable 6.2 – Experiment scenarios, prototypes and report Iteration 1, 2011.

[21] CONNECT Consortium. Deliverable 5.3 – Consolidated dependability framework, 2012.

[22] CONNECT Consortium. Deliverable 6.3 – Experiment scenarios, prototypes and report - Iteration 2, 2012.

[23] A. Correia and F. B. e Abreu. Model-driven service level management. In *4th International Conference on Autonomous Infrastructure, Management and Security, AIMS 2010, Zurich, Switzerland, June 23-25, 2010. Proceedings*, volume 6155 of *Lecture Notes in Computer Science*, pages 85–88. Springer.

[24] A. Correia, F. B. e Abreu, and V. Amaral. Slalom: a language for sla specification and monitoring. *CoRR*, abs/1109.6740, 2011.

[25] G. Cugola and A. Margara. TESLA: a formally defined event specification language. In *Proceedings of DEBS*, pages 50–61, 2010.

[26] J. Daniel, B. Traverson, and S. Vignes. A qos meta model to define a generic environment for qos management. In C. Linnhoff-Popien and H.-G. Hegering, editors, *Trends in Distributed Systems: Towards a Universal Service Market*, volume 1890 of *Lecture Notes in Computer Science*, pages 334–339. Springer Berlin Heidelberg, 2000.

[27] A. Finne. Towards a quality meta-model for information systems. *Software Quality Control*, 19(4):663–688, Dec. 2011.

[28] V. Forejt, M. Kwiatkowska, D. Parker, H. Qu, and M. Ujma. Incremental runtime verification of probabilistic systems. In S. Qadeer and S. Tasiran, editors, *Proc. 3rd International Conference on Runtime Verification (RV'12)*, volume 7687 of *LNCS*, pages 314–319. Springer, 2012.

[29] S. Gilmore, L. Gönczy, N. Koch, P. Mayer, M. Tribastone, and D. Varró. Non-functional properties in the model-driven development of service-oriented systems. *Software & Systems Modeling*, 10:287–311, 2011.

[30] N. Guelfi. A formal framework for dependability and resilience from a software engineering perspective. *Central European Journal of Computer Science*, 1:294–328, 2011.

[31] W. K. Hastings. Monte Carlo samping methods using Markov chains and their applications. *Biometrika*, pages 97–109, 1970.

[32] E. A.-S. Hussein, H. Abdel-wahab, and K. Maly. HiFi: A New Monitoring Architecture for Distributed Systems Management. In *Proceedings of ICDCS*, pages 171–178, 1999.

[33] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[34] B. Kitchenham. Procedures for performing systematic reviews. In *Joint Technical Report, Keele University Technical Report TR/SE-0401 and NICTA Technical Report 0400011T.1*, 2004.

[35] M. Kwiatkowska, D. Parker, and H. Qu. Incremental quantitative verification for Markov decision processes. In *Proc. DSN-PDS'11*, pages 359–370. IEEE, 2011.

[36] M. Kwiatkowska, D. Parker, H. Qu, and M. Ujma. On incremental quantitative verification for probabilistic systems. In *Proc. Higher-Order Workshop on Automated Runtime verification and Debugging (HOWARD-60)*. Springer, 2011. To appear.

[37] M. Mansouri-Samani and M. Sloman. GEM: a generalized event monitoring language for distributed systems. *Distributed Systems Engineering*, 4(2):96–108, 1997.

[38] A. D. Marco, F. Lonetti, and G. D. Angelis. Property-driven software engineering approach. In *Proceedings of Fifth International Conference on Software Testing, Verification and Validation*, pages 966–967, 2012.

[39] A. Miede, N. Nedyalkov, C. Gottron, A. Konig, N. Repp, and R. Steinmetz. A generic metamodel for it security attack modeling for distributed systems. In *International Conference on Availability, Reliability, and Security*, pages 430 –437, feb. 2010.

[40] J. J. Muhammad Qaiser Saleem and M. F. Hassan. Secure business process modelling of soa applications using umlsoasec. *International Journal of Innovative Computing, Information and Control*, 8:2729–2746.

[41] PMM. http://labse.isti.cnr.it/tools/pmm.

[42] O. S. Ramón, F. Molina, J. G. Molina, and J. A. T. Álvarez. Modelsec: A generative architecture for model-driven security. *J. UCS*, 15(15):2957–2980, 2009.

[43] A. Rodríguez, E. Fernández-Medina, J. Trujillo, and M. Piattini. Secure business process model specification through a uml 2.0 activity diagram profile. *Decis. Support Syst.*, 51(3):446–465, June 2011.

[44] R. Rubinstein and W. Davidson. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1:129–190, 1999.

[45] R. Saadi, M. Rahaman, V. Issarny, and A. Toninelli. Composing trust models towards interoperable trust management. In I. Wakeman, E. Gudes, C. Jensen, and J. Crampton, editors, *Trust Management V*, volume 358 of *IFIP Advances in Information and Communication Technology*, pages 51–66. Springer Berlin Heidelberg, 2011.

[46] M. Saeki and H. Kaiya. Measuring characteristics of models and model transformations using ontology and graph rewriting techniques. In *Evaluation of Novel Approaches to Software Engineering*, volume 69 of *Communications in Computer and Information Science*, pages 3–16. 2010.

[47] M. Q. Saleem, J. Jaafar, and M. F. Hassan. Security modeling of soa system using security intent dsl. In *ICSECS (3)*, volume 181 of *Communications in Computer and Information Science*, pages 176–190. Springer, 2011.

[48] Y. Shi and R. Eberhart. A modified particle swarm optimization. In *IEEE International Conference on Evolutionary Computation*, pages 69–73. IEEE, 1995.

[49] S. Supakkul and L. Chung. A uml profile for goal-oriented and use case-driven representation of nfrs and frs. *Software Engineering Research, Management and Applications, ACIS International Conference on*, 0:112–121, 2005.

[50] J. Trujillo, E. Soler, E. Fernández-Medina, and M. Piattini. A uml 2.0 profile to define security requirements for data warehouses. *Comput. Stand. Interfaces*, 31(5):969–983, Sept. 2009.

[51] A. I. F. Vaz and L. N. Vicente. A particle swarm pattern search method for bound constrained global optimization. *J. Global Optimization*, 39(2):197–219, 2007.

[52] N. Zannone. The si* modeling framework: metamodel and applications. *International Journal of Software Engineering and Knowledge Engineering*, 19(5):727–746, 2009.

[53] L. Zhu and Y. Liu. Model driven development with non-functional aspects. In *Proceedings of the 2009 ICSE Workshop on Aspect-Oriented Requirements Engineering and Architecture Design*, EA '09, pages 49–54, Washington, DC, USA, 2009. IEEE Computer Society.

[54] D. Zimmer and R. Unland. On the semantics of complex events in active database management systems. In *Proceedings of the 15th International Conference on Data Engineering*, ICDE '99, pages 392–399, 1999.

[55] G. Zoughbi, L. Briand, and Y. Labiche. Modeling safety and airworthiness (rtca do-178b) information: conceptual model and uml profile. *Softw. Syst. Model.*, 10(3):337–367, July 2011.

6 Appendix

An important objective of this last year of activity within WP5 has been to produce archival papers reporting the main results from the research and prototype development carried out in CONNECT.

In the following of this appendix, we include some papers that are in various stage of maturity.

P1 **Meta-Modeling of Non-Functional-Properties**, by Francesca Lonetti (ISTI-CNR), Antinisca Di Marco (Univaq), Antonia Bertolino (ISTI-CNR), Antonino Sabetta (SAP, formerly ISTI-CNR), to be submitted to peer-reviewed journal.

P2 **An Approach to Adaptive Dependability Assessment in Dynamic and Evolving Connected Systems**, by Antonia Bertolino (ISTI-CNR), Antonello Calabró (ISTI-CNR), Felicita Di Giandomenico ISTI-CNR, Nicola Nostro (ISTI-CNR), to appear, paper accepted for publication in International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS), 2013.

P3 **On-the-Fly Dependable Mediation between Heterogeneous Networked Systems**, by Antonia Bertolino (ISTI-CNR), Antonello Calabró (ISTI-CNR), Felicita Di Giandomenico ISTI-CNR, Nicola Nostro (ISTI-CNR), Paola Inverardi (Univaq), Romina Spalazzese (Univaq), to appear, paper accepted as book chapter in M. Jose Escalona, J. Cordeiro, and B. Shishkov (Eds.): ICSOFT 2011, CCIS 303, pp.20-37, 2012 (Springer-Verlag book collecting extended versions of best papers at ICSOFT 2011).

P4 **Enhanced Connectors Synthesis to Address Functional, Performance, and Dependability Aspects**, by Romina Spalazzese (Univaq), Nicola Nostro (University of Florence, formerly (ISTI-CNR)), Felicita Di Giandomenico (ISTI-cNR), Paola Inverardi (Univaq), to be submitted to peer-reviewed journal.

P5 **Networks of Heterogeneous and Dynamic Interoperable Systems: An Approach to Enhance Dependability and Performance Properties**, by Felicita Di Giandomenico (ISTI-CNR), Massimiliano Itria (ISTI-CNR), Paolo Masci (Queen Mary University of London, formerly ISTI-CNR) and Nicola Nostro (University of Florence, formerly ISTI-CNR), to be submitted to peer-reviewed journal.

Meta-Modeling of Non-Functional Properties

Francesca Lonetti

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", CNR, Pisa Italy

Antinisca Di Marco

University of L'Aquila, L'Aquila, Italy

Antonia Bertolino

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", CNR, Pisa Italy

Antonino Sabetta

SAP Research, Sophia Antipolis, France

Abstract

In Model Driven Engineering (MDE) non-functional properties permeate all life-cycle. We present a generic, comprehensive and flexible Property Meta-Model (PMM) for defining non-functional properties spanning over dependability, performance and security. PMM allows for specifying metrics and provides a complex events specification language that takes into account and enhances the composition operators of existing events specification languages. We give examples of using PMM for modeling different properties and in different phases of a MDE process. Furthermore, to evaluate the comprehensiveness and flexibility of PMM with respect to similar approaches, we include results from a systematic survey on MDE approaches dealing with non-functional properties, metrics and complex events.

1. Introduction

Software production has gradually acquired an industrial character, relying more and more on automated methods and tools. An important role in this evolution is played by the model-driven engineering (MDE) paradigm; by emphasizing the role of models, now considered first-class artifacts of software development process, MDE contributes to ensure both functional and

non-functional properties, achieving more predictable results with reduced effort.

Non-functional properties, such as performance, reliability, and security, are known to require special care when composing complex systems out of pre-existing modules or services (possibly coming from different providers). Even when composing perfectly functional components, it is hard to ensure that the resulting system will exhibit an acceptable level of performance (resp.: reliability, security) because of the *emergent* behavior inherent in the composition as such.

This problem is exacerbated by the fact that dynamic reconfiguration is being more and more commonly advocated as a means to cope with (and react to) unforeseen changes in the context, such as workload spikes, unpredictable failures, or attempts to unauthorized use.

Assuring non-functional properties becomes more challenging when systems are dynamically integrated via on-the-fly composition of independently developed services or components, such as in service-oriented computing or in the CONNECT architecture [50].

To address these challenges, non-functional properties must be addressed in all the phases of the software lifecycle process, spanning requirement specification, design, and development, and extending to system deployment and execution monitoring.

In such cases, tools for runtime monitoring, analysis and enforcement of non-functional properties must be adopted to continuously ensure that the emerging behaviour is compatible with the constraints imposed on non-functional characteristics, e.g., in service-level agreements (SLAs).

While it is natural that different tools can be specific to a phase of the lifecycle and therefore they can employ a specific modelling notation, the general problem spans all the phases of the lifecycle and therefore can only be effectively tackled by coordinating the models used in each of them. Following the principle of MDE, this can be achieved by devising a core conceptual specification of non-functional requirements, defined as a meta-model, from which other specialized models can be obtained by automated model-to-model transformations.

For some time, the greatest attention of MDE research has been devoted to functional aspects, with non-functional aspects deferred to be considered later and possibly stuck on an already designed system. Nowadays, the community has broadly recognized the need of including non-functional aspects as early as possible in system design, so that, quoting Selic, we can move

from Model-Driven Development to Model-Driven Engineering [48]. As a consequence, a variety of approaches and tools for model-driven management, analysis, verification and assurance of non-functional requirements are being proposed. However, in search within such emerging literature for some modeling approach that we could adopt for our purposes, we noticed a lack of some generic, comprehensive and flexible notation.

By *generic* we mean a meta-model that is independent from the specific application so that it can be reused in varying domains. To achieve this feature, according to one founding principle of MDE a meta-model for non-functional properties should treat separately the two concerns of defining the elements that characterize a property and of assigning to such elements the appropriate domain-specific meanings.

By *comprehensive* we mean a meta-model that covers in uniform way dependability, performance and security related properties, as well as other more complex properties, such as trust, that might combine aspects from either of them.

Finally, by *flexible* we mean a meta-model that can express both qualitative and quantitative (a.k.a. metrics) properties, as well as simple and complex events that are the basis for establishing an observational measure of such properties.

Many valuable approaches have been proposed for modeling non-functional properties, and we devote a large portion of this paper to present a systematic review of them. As a result of this review, we could not identify a single that could satisfy all the above requirements at the same time. Therefore, we engaged in the ambitious goal of defining "yet another" Property Meta-Model (PMM) that could fill the existing gap in the area of model-driven management of non-functional requirements.

In [22], a preliminary version of PMM was briefly presented mostly focusing on the meta-model part that defines qualitative and quantitative properties, whereas in [10] we gave an overview of the complex events specification language also part of PMM. In this paper, we present the complete and finalized specification of PMM and its extensive comparison against existing approaches. The intent of such a comparison is to validate our claim that PMM is more generic, comprehensive and flexible than any other competitor approach. Therefore, in order to have a complete and sustainable validation, this comparison is made against a systematic survey of related works in the literature, which per se constitutes another original contribution of the paper.

PMM is implemented as an eCore model and is made available to the community for adoption, validation and possible extension. The meta-model is provided along with an associated editor realized as an Eclipse Plugin. In our intent, by using PMM a software developer could either find and retrieve from the editor repository the pre-built specification of a simple or complex extra-functional property or otherwise easily specify new properties and metrics of interest, using the concepts in the meta-model. The PMM compliant models can be adopted all along the software lifecycle and at runtime. To exemplify the multiple uses of PMM we include a description of how it has been applied, within the CONNECT project, for the two purposes of synthesizing a connector satisfying given extra-functional requirements, and of instructing a model-driven monitor about the extra-functional properties to be detected.

Summarising, the contribution of this paper includes:

- a generic, comprehensive and flexible Property Meta-Model (PMM) defining the abstract structure of non-functional properties that span over dependability, performance and security;
- a comprehensive and machine processable specification language (included into PMM, but also usable in isolation) that allows for defining complex events models involved into non-functional properties. The proposed language improves over existing complex events specification languages by adding new features not included in the existing languages;
- examples of use of PMM models in two different application domains that are synthesis and monitoring of non-functional properties;
- a systematic literature review of MDE approaches dealing with non-functional properties, metrics and complex events.

The paper content is structured as follows. Section 2 describes PMM while Section 3 presents some modeling examples of properties, metrics and events. In Section 4 we evaluate the proposed approach showing the model-driven support of PMM along two key activities for the development of dynamic and evolving systems that are monitoring and synthesis. Section 5 contains the comparative results of a systematic survey addressing meta-modeling approaches similar to PMM. Finally, Section 6 draws conclusions and future research direction.

2. Property Meta-Model

PMM is a generic, comprehensive and flexible meta-model specifying non-functional properties, metrics and events. The definition of properties and metrics is independent from the application domain. The defined properties represent quantitative and qualitative properties that a generic software system or its components may expose (descriptive properties), or must provide (prescriptive properties). The concepts and terms belonging to an application domain are linked to PMM via the eventType and action entities, which model a generic observed event or an atomic action, respectively, of the application domain in which the system will be used. Thus, we distinguish between a generic metric formula (represented by a metricsTemplate) and a concrete metric (i.e., the metrics concept) that is instantiated onto a specific application domain by means of the eventSet and eventType concepts. This approach makes our meta-model more generic, since a same template can be used (instantiated) in different scenarios. PMM specification includes a machine processable specification language that allows for defining complex events models involved into non-functional properties. The proposed language is specified as part of PMM, but can be used in isolation to specify events that are not necessarily tied to a property model. This specification language combines features of two existing event specification languages that are GEM [32] and Drools Fusion [24] and in addition presents new features not included in the considered languages. In particular, we have defined operators that allow for modeling a temporal relationship (as those of Drools Fusion) and operators that allow for combining simple or complex events (as those of GEM), and in addition we have identified situations of interest not covered by the operators of GEM and Drools Fusion that have been formalized through new operators.

In Figure 1 we present the structure and the main concepts of PMM. Note that for the sake of readability, this figure has been obtained as an excerpt of the complete meta-model by visualizing only the main concepts and removing several technical details. We also introduced dashed frames around the shown elements to highlight the PMM sub-meta-model they belong to. As the figure shows, such main concepts are: property, metricsTemplate, metrics, eventSet, and eventType.

The PMM meta-model has been generated as an eCore model into the Eclipse EMF framework. In particular, this meta-model has been partitioned in the following eCore models: *Core.ecore* representing a generic named el-

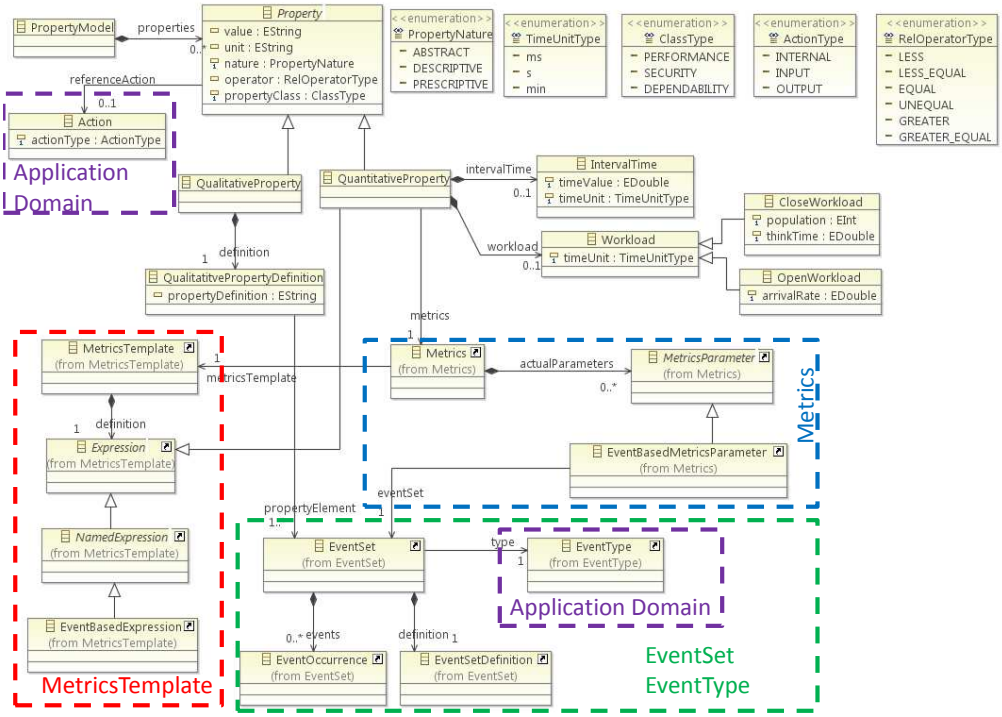


Figure 1: Key Concepts of the Property Meta-Model.

ement, *Event- Type.ecore* and *EventSet.ecore* modeling the event and the eventSet respectively, *Metrics.ecore* and *MetricsTemplate. ecore* for specifying the metrics and metricsTemplate concepts and finally the*Property.ecore* representing the property meta-model. All information about the classes defined into each eCore model are included into the corresponding *model-name.ecore* document. An editor associated to PMM has been generated as an Eclipse Plugin and contains the information of the defined eCore models. It allows for defining new model instances of the Property, Metrics, MetricsTemplate, EventType and EventSet meta-models[1].

In the following we provide the description of the proposed meta-model by giving details on the Property meta-model (in Section 2.1); Metrics and

[1]A release of the Property Meta-Model and the associated editor is available at http://labse.isti.cnr.it/tools/pmm.

MetricsTemplate meta-model (in Section 2.2); and finally on the EventSet and EventType meta-model (in Section 2.3). In the description we adopted the following typing convention: concepts of the meta-model are given in `typewriter` style (it will be italicized *`typewriter`* style for abstract classes). Values of enumerated types are generally reported in `CAPITALISED` letters, as they appear in the meta-model.

2.1. Property specification

Figure 2 reports the portion of the meta-model describing a *`Property`*. A `PropertyModel` is composed by zero (modeling the empty model) or more properties.

A *`Property`* is a *`NamedElement`* having two required/mandatory attributes and three optional ones. The required attributes are: `nature` and `propertyClass`. The `nature` attribute refers to the nature of the property that can be `ABSTRACT`, `DESCRIPTIVE`, or `PRESCRIPTIVE`, whereas the `propertyClass` can have the following values: `PERFORMANCE`, `SECURITY` and `DEPENDABILITY`.

An abstract property indicates a generic property that does not specify a required or guaranteed value for an observable or measurable feature of a system. A descriptive property represents a guaranteed/owned property of the system while a prescriptive one indicates a system requirement. In both cases, the property is defined taking into account a relational operator with a specified value. The optional attributes of *`Property`* are `value`, `unit` and `operator`. As specified in an OCL constraint (see upper note at the right-hand side of Figure 2), these attributes are not specified in case of an abstract property because, as described above, an abstract property does not specify a relation with a specific value. They are specified only for the descriptive and prescriptive properties. The `value` attribute indicates a value associated to the property, the `unit` attribute indicates its unit of measure whereas the `operator` one models a relational operator (one of the operators listed in the `RelOperatorType` enumeration).

The `QuantitativeProperty` can have a `Workload` and/or an `IntervalTime`. The former is mandatory for performance properties, while the latter is mandatory for dependability ones. Both are not specified for an abstract property. The `Workload` is a meta-model element that can be open or close and has the `timeUnit` attribute specified according to one of the time units listed in the `TimeUnitType` enumeration. The `IntervalTime` is an element

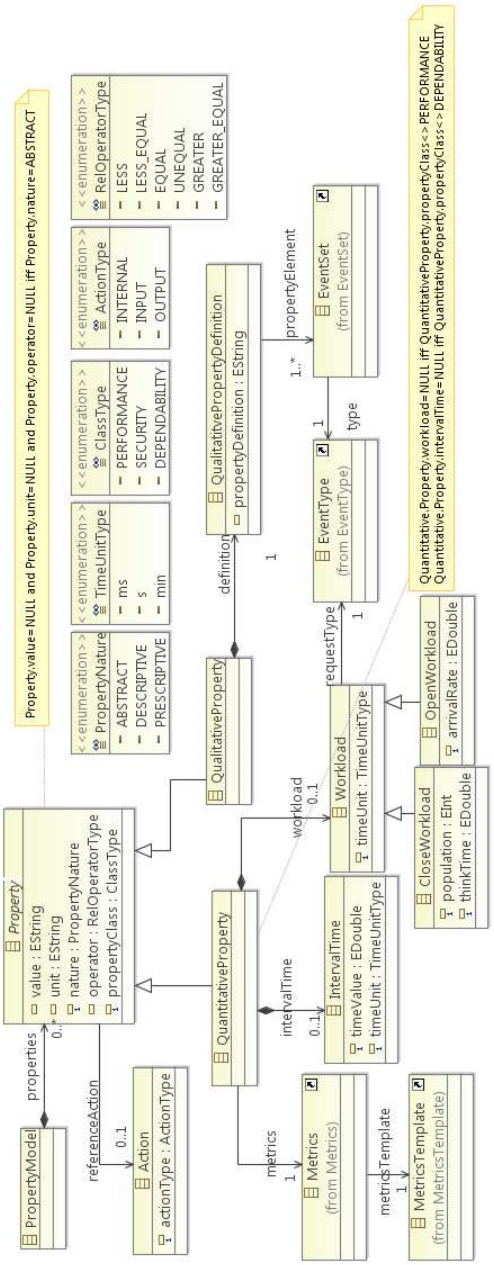


Figure 2: Property.

that has the `timeUnit` and the `timeValue` attributes, needed to specify the interval of time it represents.

A ***`Property`*** can have an associated `Action` representing the execution of a behavior or operation of the system. The `Action` element models a system behavior as atomic without capturing its complexity or details. An `Action` has the `actionType` attribute that refers to the type of the action and can have the following values: `INTERNAL`, `INPUT` and `OUTPUT`.

To clarify the above concepts we report below some properties examples:

[Property1:] Ability to provide a service according to given time requirements.

This is an abstract property since it does not deal with a claimed or guaranteed specified time feature. This property is also a quantitative property having a performance class because it is about a measurable performance dimension (time). In this case, the `value`, `unit` and `operator` attributes of ***`Property`*** and the `Workload` and `IntervalTime` ones of
`QuantitativeProperty` are not specified.

[Property2:] The service S in average responds in 3 ms in executing the e_1 operation when it is subject to an open workload with arrivalRate *of 10 e_2 operations per time unit.*

This is a descriptive property asserting that the service S guarantees in average a time response having a value of 3 ms in executing the e_1 operation, with a workload of 10 e_2 concurrent operations. As *Property1*, this one is also a quantitative property having a performance class because it is about a measurable performance dimension (time). In this case, the `value` attribute is equal to 3, the `unit` measure is ms and the specified `operator` is `EQUAL`. The associated `OpenWorkload` is characterized by the `arrivalRate` equal to 10 while `IntervalTime` is not specified.

[Property3:] The service S in average must respond in 3 ms in executing the e_1 operation when it is subject to an open workload with arrivalRate *of 10 e_2 operations per time unit.*

This is a prescriptive property because it specifies a required time response. As *Property1* and *Property2*, this is also a quantitative property having a performance class. The `value` attribute, the `unit` mea-

sure, the specified `operator` and the `Workload` are equal to those of *Property2*.

In the above examples, operations S, e_1 and e_2 are application-dependent and their semantics can be given once the application domain has been fixed.

2.2. Metrics specification

The Metrics meta-model describes two main concepts: the `MetricsTemplate` and the `Metrics`.

To understand the difference between them, let us make an intuitive example considering the response time. Intuitively, we understand that the response time of a system, for a given operation, is the time interval between submitting the operation request and receiving the response from the system. More precisely, this concept could be defined either as the duration of a complex operation or as the difference between the timestamps of two atomic operations (request and response) considered as having no duration. This example thus shows two different ways to specify the metric of response time. We call each of such specifications a metricsTemplate. Hence a metric (response time) can refer to one or more specifications (metricsTemplates).

More precisely, we can have the two following metricsTemplates for response time:

1. response time of the E operation = DURATION(E) where E is the operation of which we want to measure the response time.
2. response time of the E operation = timestamp(E2)-timestamp(E1), where E1 and E2 represent the starting and the ending operations of E, respectively.

Moreover, as shown in 1) and 2) above, a metricsTemplate is defined upon a generic set of events/operations, that is not coupled with a particular application domain. This generic part of the definition is specialized by the metric that instantiates those general concepts (templateParameters) with application-based events/operations (metricsParameters). In 1) there is a templateParameter that is operation E, in 2) the templateParameters are E, E1 and E2.

Figure 3 reports the meta-model portion describing the `MetricsTemplate` concept. A `MetricsTemplateModel` defines one or more `MetricsTemplate`s, each having a `dimension` indicating the type of the value defined by the metrics template (e.g., a TIMEd value, a PERCENTAGE, and so on). A

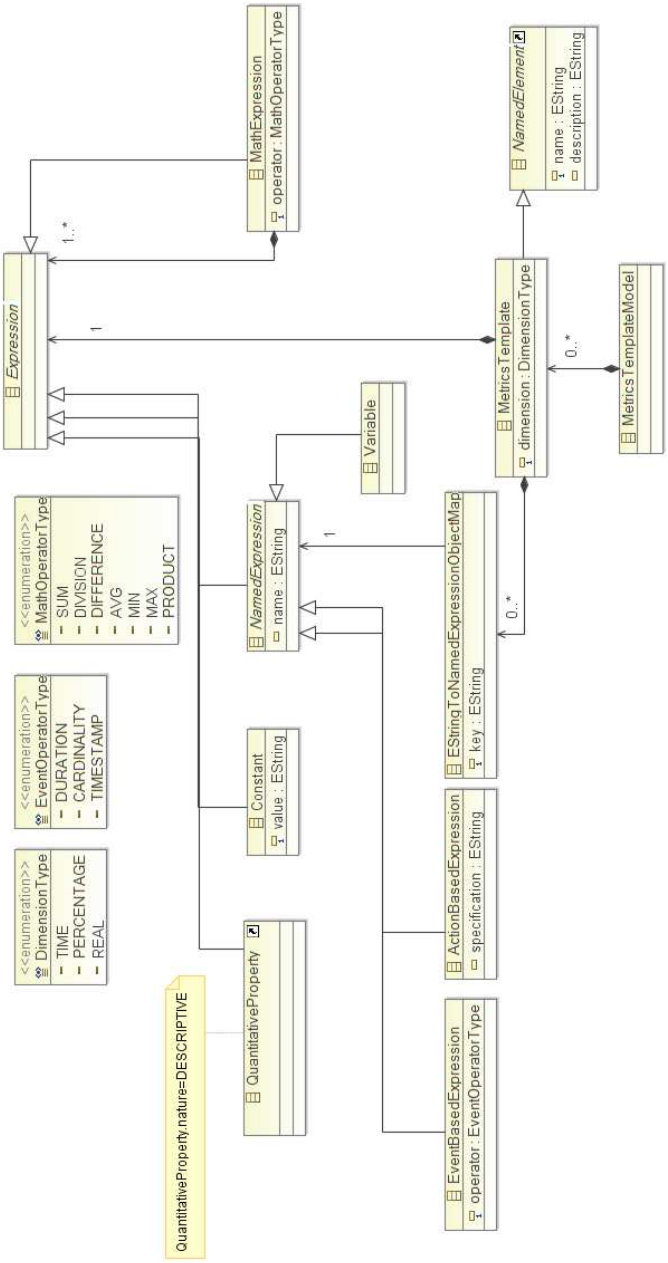


Figure 3: MetricsTemplate.

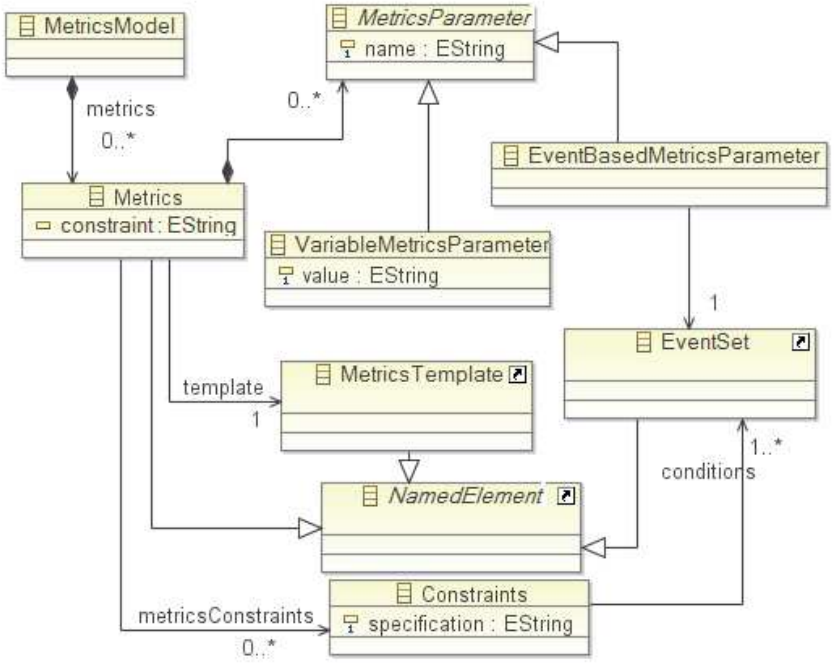


Figure 4: Metrics.

`MetricsTemplate` is a ***`NamedElement`*** containing the ***`Expression`*** describing the mathematical definition of the template.

The ***`Expression`*** could be:

- a `MathExpression`, this meta-class allows the definition of a complex mathematical expression by nesting one or more operands that are in turn other ***`Expression`***s. The `MathExpression` has an attribute, `operator`, that specifies the `MathOperatorType`. We define so far as possible operators the ones in the `MathOperatorType` enumeration. This enumeration can be extended if necessary.

- a `QuantitativeProperty`, in this case the ***`Expression`*** can be a `QuantitativeProperty` already described. To be used as the operand of an expression, it must have a descriptive nature as set by the annotated OCL constraint.

- a `Constant` that has a `value` attribute of `EString` type indicating the specific value the constant refers to.

- a *NamedExpression*. This element can represent:
  - a canonical Variable.
  - an ActionBasedExpression, that represents a simple action or a sequence of actions that, whenever executed, reports a value. An example could be a sort of get method that reports some information managed by some component of a software system.
  - an EventBasedExpression, that represents expressions based on events or observational behavior. In this case the name represents the observable, simple or complex, event/behavior the operator applies to. As set of EventOperatorType we define DURATION for a complex event indicated in the name attribute, TIMESTAMP for a simple one, and CARDINALITY for the type of event indicated in name attribute. While the first and second operators are applied to single occurrences of the indicated event type, the third one is applied to the whole set of event occurrences observed at a given time instant.

The MetricsTemplate finally contains zero or more template parameters. These template parameters are
EString keys exposed by the template and linked to *NamedExpression* by means of the
EStringToNamedEspressionObjectMAp concept.

Figure 4 reports the meta-model portion describing the Metrics concept. A MetricsModel defines zero or more Metrics. A Metrics is a *NamedElement* that refers to a MetricsTemplate, actualizes the template parameters by means of the *MetricsParameter*s (if the
MetricsTemplate has no formal parameters, Metrics exposes zero *MetricsParameter*s), and contains zero or more Constraints.

EventBasedMetricsParameter is a *MetricsParameter* actualizing the EventBasedExpression based template parameters. To this purpose, it refers to the EventSet describing the application based event definition and the relative occurrences. VariableMetricsParameter instead is a *MetricsParameter* actualizing the Variable template parameter indicating the corresponding value.

Finally, Constraints defines some condition (stored in the specification attribute) on the EventSet involved in the Metrics that must be satisfied.

Such constraints in general allow for making a correspondence among event occurrences belonging to different types of `EventSet` in case the `EventType`s in the `EventSet`s have some `EventTypeParameter`s.

2.3. Complex Events Specification

The portion of the meta-model describing the events includes the `EventType` and the `EventSet` concepts.

As shown in Figure 5, the `EventTypeModel` is composed by zero or more `EventType` elements, each one modeling an observable system behavior that is the manifestation/expression that something of interest (system action or activity) happens. It can be a primitive/simple event, representing the lowest observable system activity, or a composite/complex event, representing a combination of primitive or other composite events. More precisely, an `EventType` has one or more parameters, one constraint and is composed by one or more `ComplexEvent` that combines primitive and other composite events by means of the operators defined in `OperatorType`. The required `compositionOrder` attribute represents the order of the events in the composition and can take one of the values listed in the `Ordering` enumeration. In Table 1 we describe all the operators defined in `OperatorType` and their parameters.

The operators defined in PMM include all temporal operators of Drools Fusion (*after*, *before*, *coincides*, *during*, *finishes*, *finished by*, *includes*, *meets*, *met by*, *overlaps*, *overlapped by*, *starts*, *started by*) and improve their definition distinguishing operators with the same type but different number of parameters. We distinguish for instance *During_2p* and *During_4p* operators representing the same *During* operator with two and four parameters respectively. The PMM complex events language also includes the basic composition operators defined in GEM (for instance *or* and *interleaving*) and defines new operators (for instance *not*) not defined in the considered languages. We refer to Section 5 for a comparison of PMM complex events specification language with GEM and Drools Fusion languages, whereas in [10] we provide a detailed mapping of PMM complex events operators on those of GEM and Drools Fusion.

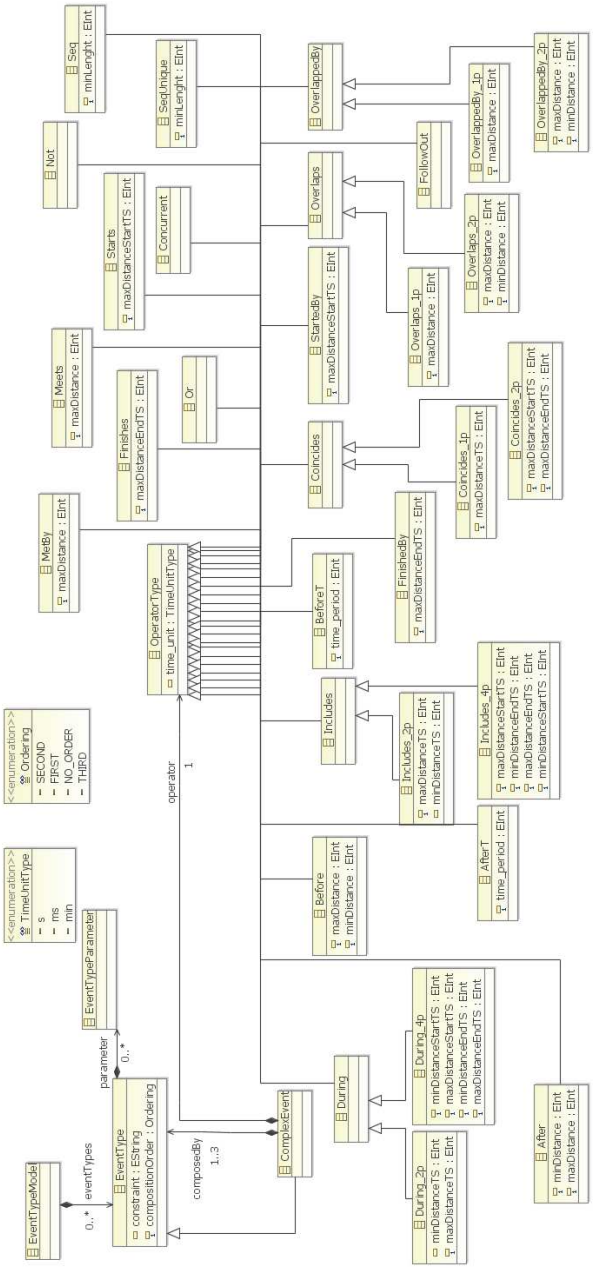


Figure 5: EventType

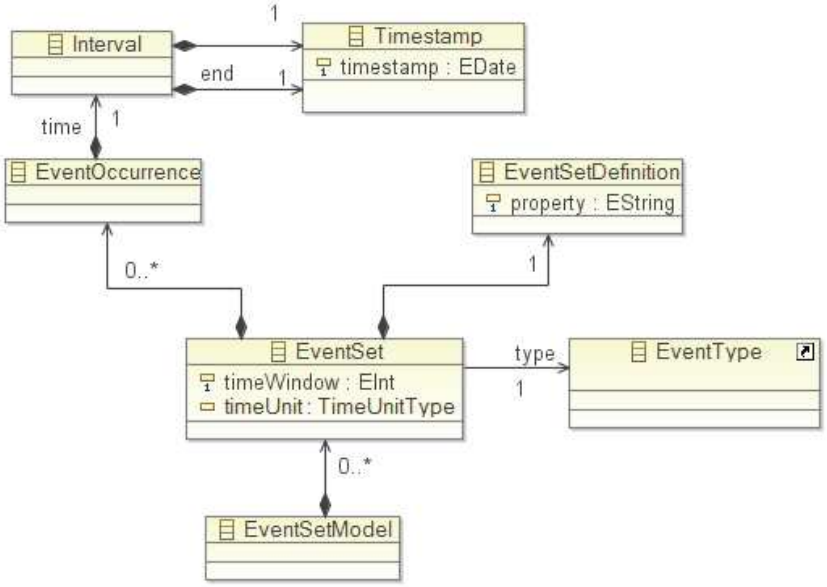


Figure 6: EventSet.

Table 1: Event composition operators

Operator	Description	Parameters	Parameter meaning
After (Before)	involves two events and occurs when the current event (e1) happens after (before) the correlated event (e2)	*minDistance*	min time distance between e2 (e1) finishing and e1 (e2) starting
		maxDistance	max time distance between e2 (e1) finishing and e1 (e2) starting
AfterT (BeforeT)	involves one event and occurs when the event happens after (before) the specified time-period	*time_period*	time period before (after) that the event occurs

Coincides_1p	occurs when both events happen at the same time, the start timestamps of both e1 and e2 are the same AND the end timestamps of both e1 and e2 are also the same	*maxDistanceTS*	max distance between e1 and e2 start timestams and max distance between e1 and e2 end timestamps
Coincides_2p	occurs when both happen at the same time, the start timestamps of both e1 and e2 are the same AND the end timestamp of both e1 and e2 are also the same	*maxDistanceStartTS*	max distance between e1 and e2 start timestamps
		maxDistanceEndTS	max distance between e1 and e2 end timestamps
Concurrent	involves two events and occurs when both events happen irrespective of their order	-	-
During_2p	occurs when the current event (e1) starts after the correlated event (e2) and finishes before it	*maxDistanceTS*	maximum distance between the start timestamp of e1 and e2 and the maximum distance between the end timestamp of e1 and e2
		minDistanceTS	min distance between the start timestamp of e1 and e2 and the maximum distance between the end timestamp of e1 and e2
During_4p	occurs when the current event (e1) starts after the correlated event (e2) and finishes before it	*maxDistanceStartTS*	maximum distance between the start timestamp of e1 and e2
		minDistanceStartTS	minimum distance between the start timestamp of e1 and e2
		maxDistanceEndTS	maximum distance between the end timestamp of e1 and e2

		minDistanceEndTS	minimum distance between the end timestamp of e1 and e2
Finishes	occurs when the current event (e1) starts after the correlated event (e2) but both events end at the same time	*maxDistanceEndTS*	maximum distance between the end timestamp of e1 and e2
FinishedBy	occurs when the current event (e1) starts before the correlated event (e2) but both events end at the same time	*maxDistanceEndTS*	maximum distance between the end timestamp of e1 and e2
FollowOut	involves three events and occurs when the first event is followed by the second event and without the occurrence of the third event	-	-
Includes_2p	occurs when the correlated event (e2) starts after the current event (e1) and finishes before it	*maxDistanceTS*	maximum distance between the start timestamp of e1 and e2 and the maximum distance between the end timestamp of e1 and e2
		minDistanceTS	minimum distance between the start timestamp of e1 and e2 and the minimum distance between the end timestamp of e1 and e2
Includes_4p	occurs when the correlated event (e2) starts after the current event (e1) and finishes before it	*maxDistanceStartTS*	maximum distance between the start timestamp of e1 and e2
		minDistanceStartTS	minimum distance between the start timestamp of e1 and e2
		maxDistanceEndTS	maximum distance between the end timestamp of e1 and e2
		minDistanceEndTS	minimum distance between the end timestamp of e1 and e2

Overlaps_1p	occurs when the current event (e1) starts before the correlated event (e2) starts and finishes after the correlated event starts, but before the correlated event finishes.	*maxDistance*	maximum distance between the start timestamp of the e2 and the end timestamp of e1
Overlaps_2p	occurs when the current event (e1) starts before the correlated event (e2) starts and finishes after the correlated event starts, but before the correlated event finishes	*maxDistance*	maximum distance between the start timestamp of the e2 and the end timestamp of e1
		minDistance	minimum distance distance between the start timestamp of the e2 and the end timestamp of e1
OverlappedBy	the correlated event (e2) starts before the current event (e1) starts and finishes after the current event starts, but before the current event finishes	*maxDistance*	maximum distance between the start timestamp of e1 and the end timestamp of the e2
OverlappedBy_2p	the correlated event (e2) starts before the current event (e1) starts and finishes after the current event starts, but before the current event finishes	*maxDistance*	maximum distance between the start timestamp of e1 and the end timestamp of the e2
		minDistance	minimum distance between the start timestamp of e1 and the end timestamp of e2

Meets (MetBy)	occurs when the current event (e1) finishes (starts) when the correlated event (e2) starts (finishes)	*maxDistance*	max distance between e1 (e2) end timestamp and e2 (e1) start timestamp
Starts (StartedBy)	occurs when the current event e1 (the correlated event e2) finishes before e2 (e1)	*maxDistanceStartTS*	maximum distance between the start timestamp of e1 and e2
Not	the specified event doesn't happen	-	-
Or	involves two events and occurs when one of the two events happens	-	-
Seq	involves one event and occurs when there is a sequence of occurrences of it	*minLenght*	minimum length of the sequence
SeqUnique	involves one event and occurs when there is a sequence of occurrences of it without duplicate occurrences	*minLenght*	minimum length of the sequence

Figure 6 reports the portion of the meta-model describing the `EventSetModel`. The `EventSetModel` has zero or more `EventSet`. An `EventSet` represents a set of event instances that refer to an `EventType`. The `property` of an `EventSetDefinition` identifies all observable events that have to be included in the *EventSet*. An `EventSet` has zero or more `EventOccurrence`s representing the observable events that the `EventSet` contains.

An `Event_Occurrence` refers to an `Interval` that represents the time range in which the observable event occurs. Each `Interval` has two associated `Timestamp`s indicating its starting and ending date respectively. These `Timestamp`s are equal in case of atomic/instantaneous event occurrences.

The event occurrences collected into an `EventSet` define observable system behaviors modeled by qualitative properties or represent the parameters

upon which the metrics are defined for measuring quantitative properties.

3. PMM Modeling Examples

In this section we show some modeling examples using PMM. We consider a simple application scenario of a security guards command patrolling an area [2]. As depicted in Figure 7, we consider here the case of a critical situation in which when a threat is detected, the Guard Control Center sends immediately an alert request to the Commander controlling the selected critical area. The Commander first notifies the Control Center that he/she takes in charge the request and then sends an *EmergencyAlert* message to the guards of the controlled area. The guards are equipped with ad hoc handheld devices that are connected to the Commander and registered to receive commands and documents. On correct receipt of the alert, each guard device automatically sends an *eAck* message to the Commander.

We provide the models of two non-functional aspects of interest for the above scenario. Specifically, we show in Section 3.1 the models of a DEPENDABILITY property (coverage property) whereas in Section 3.2 we model a security contract among the involved parts. We further refer to [41] for examples of PERFORMANCE property models.

3.1. Coverage *Property*

We show how to model the following required *coverage* property: *the average percentage of guard devices that are reached in 10 seconds by the alert message must be greater than* 70%. This means that, after 10 seconds from the *EmergencyAlert*, at least 70% of guard devices reply with an *eAck*. The model for this property is shown in Figure 8. This property is a *PRESCRIPTIVE DEPENDABILITY* property (i.e. a dependability requirement) specifying that the *CoverageReachingGuard* metrics must be *GREATER* than *70%* after an *IntervalTime* of *10 seconds* starting from the *EmergencyAlert* event occurrence.

The *CoverageReachingGuard* metrics in Figure 10 actualizes the corresponding *Average Coverage Metrics Template* (see Figure 9) by linking to the *TemplateParameter*s the corresponding *EventSet*s.

[2] This example is excerpted from one of the demonstration scenarios used in the CONNECT Project [15].

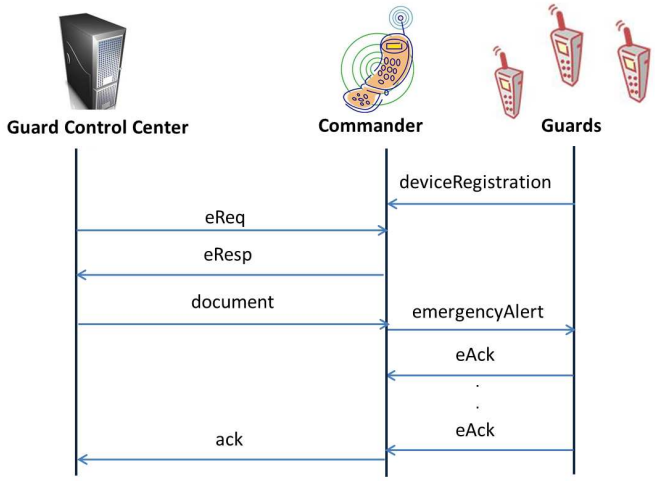


Figure 7: An overview of the Guards scenario.

The average coverage represents a *PERCENTAGE* measure defined as average of the division among the *CARDINALITY* of two sets of instances of two types of events, named x and y. Finally, the template exposes two *templateParameters*, e linked to x, and e_3 linked to y (see Figure 9). We recall that the template is generic and can be used in other scenarios, this shows the flexibility of the proposed meta-model.

Figures 11 and 12 report the model for the e and e_3 event sets, respectively. e_3 *EventSet* refers to *deviceRegistration EventType* by introducing the following condition: there are no duplication in the occurrences of the

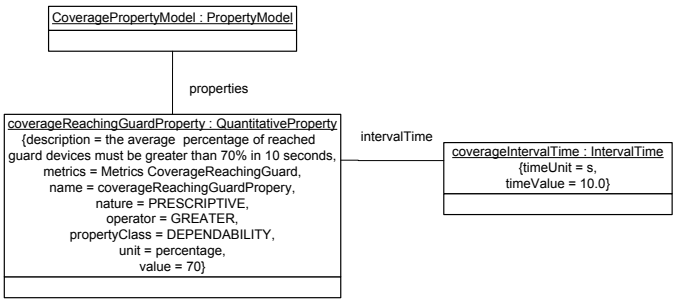


Figure 8: PMM model for Coverage Property.

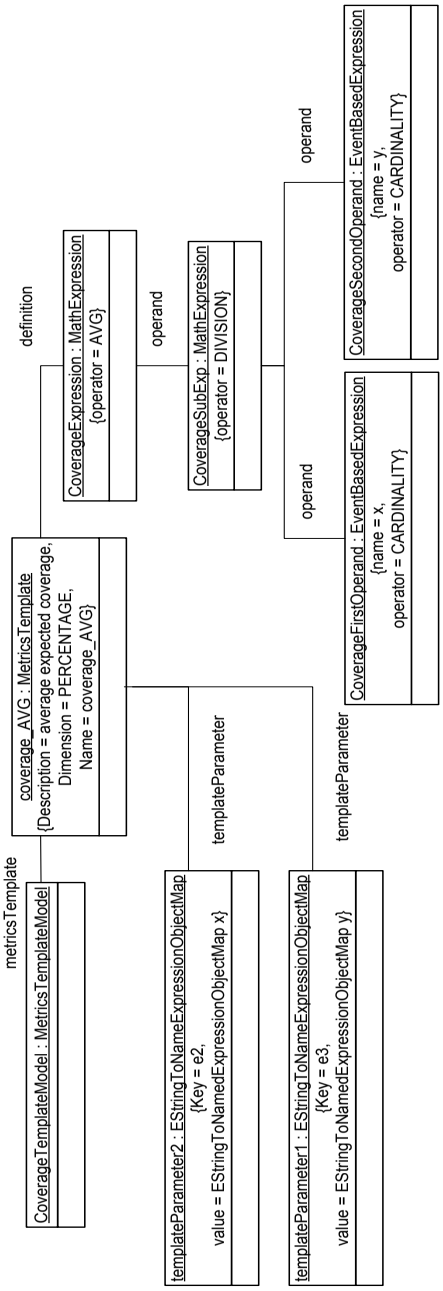


Figure 9: PMM model for Average Coverage Metrics Template.

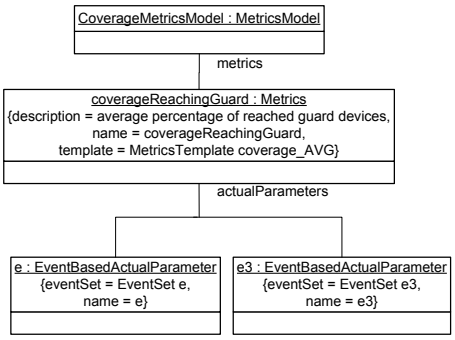


Figure 10: PMM model for Average Coverage Metrics.

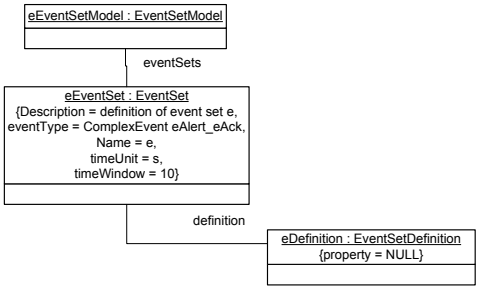


Figure 11: PMM model for e EventSet

deviceRegistration event, that is there are not two different occurrences of the *deviceRegistration* coming from the same device (i.e., events having the same *IDg* value).

The *deviceRegistration EventType* presented in Figure 13, has a simple event definition since it corresponds to a message directly observable from the system, namely interface operation. The signature of the interface operation is *deviceRegistration(IDg)* as specified in the *EventTypeSpecification* element. This comes from the ontology created for the scenario. The *EventType* has a *parameter* that is the formal parameter of the interface operation.

The *e EventSet* in Figure 11 refers to *eAlert_eAck EventType* without introducing additional conditions.

The *eAlert_eAck EventType*, shown in Figure 14, is a complex event representing the *EmergencyAlert* with its related *eAck* from the guards. The

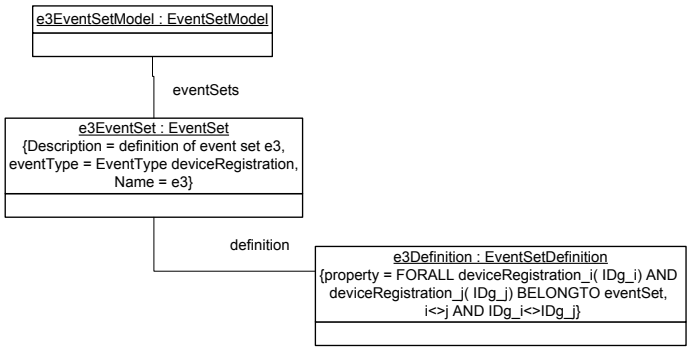


Figure 12: e_3 Actual Parameter in the CoverageReachingGuards Metrics Model

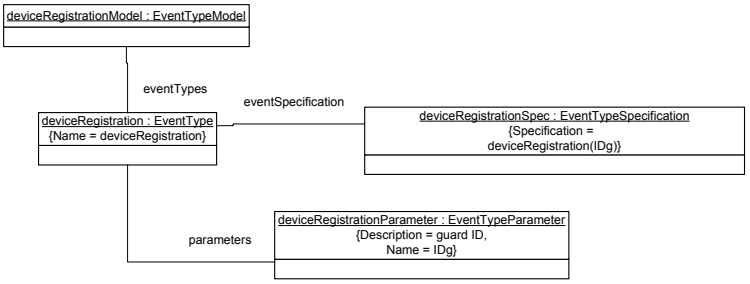


Figure 13: PMM model for Guard Device Registration

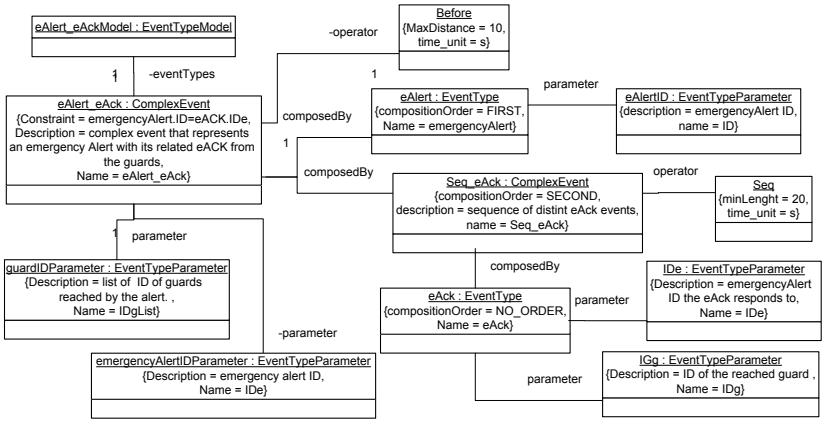


Figure 14: Sequence of Ack for an Alert

Constraint attribute defines the related condition that imposes that all the *eAck.IDe* must be equal to the *emergencyAlert.ID*. *eAlert_eAck* has a *Before* operator with *maxDistance* parameter equal to 10. This operator is applied to *eAlert* simple *EventType* and to *Seq_eAck* complex *EventType* representing respectively the former and the latter events to which the *Before* operator is applied. The *eAlert EventType* has the *eAlertID* parameter representing the *EmergencyAlert* ID the sequence refers to. The *Seq_eAck* is another *ComplexEvent* type with *Seq* operator (see Table 1). It is composed by *eAck EventType*, with two parameters: *IDg* that is the ID of the reached guard and *IDe* that is the *EmergencyAlert* ID the *eAck* responds to. In this case the event *compositionOrder* is *NO_ORDER*. The *eAlert_eAck EventType* has two parameters: the *EmergencyAlert* ID (namely IDe) the sequence refers to, and the list of guards messages acknowledging the alert (namely, IDgList).

3.2. Security Contract

For security analysis and enforcement, in the CONNECT project we have applied the Security-by-Contract paradigm (S×C) [23]. This approach (which takes inspiration from the *programming-by-contract* paradigm) works on the assumption that an application/service exposes a contract of its security guarantees. Hence, a user of the application/service before adopting it can check the contract against the required policy. If the contract does not satisfy the policy, S×C will apply a suitable enforcement mechanism.

Using PMM, we have been able to model the *contract* eventType within the scenario of the patrolling guards depicted in Figure 7. In particular, the *contract* agreed among the involves parties (the Guard Control Center, the Commander and the Guards) states that to guarantee a secure interaction, a specified sequence of events must be observed.

The model for the *contract* is given in Figure 15. In textual form, that we report to help understanding the model, the *contract* could be specified as follows:

```
((SelectedArea Before eReq) Before
(eResp[IDc] Before areaSelected)) Before
((uploadData Before emergencyAlert) Before
(SeqUnique(timeout Or eAck[IDg])
Before uploadSuccess)),
```

where *IDg* and *IDc* represent the guard and the commander ID, respectively. Such *IDs* are parameters for the related events in the *contract* model.

More in detail:

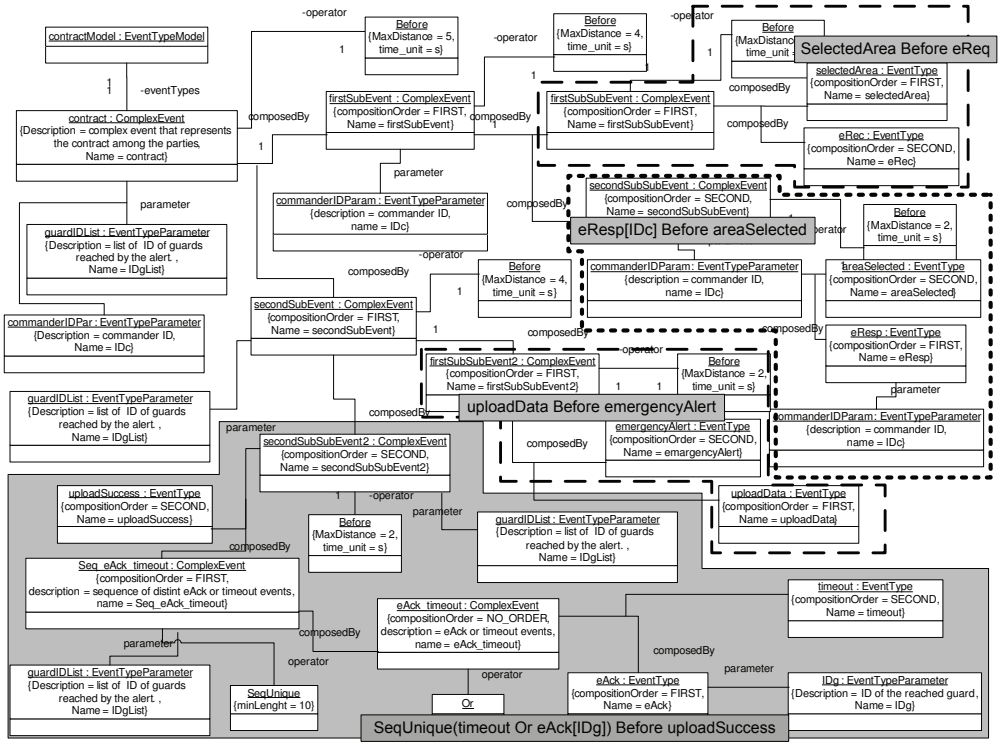


Figure 15: The Contract EventType

1. the Control Center first selects an area of interest and then sends a request *eReq* to the Commander of the area;
2. the Commander first responds with an *eResp* message and then takes in charge the selected area;
3. the Control Center sends some document to the Commander and then this latter sends an emergency alarm to a group of Guards;
4. the Commander waits for the acknowledge of receipt from the Guards until a timeout expires;
5. the Commander notifies the Control Center that the Alarm has been successfully handled.

In Figure 15 we report, for each portion of the *contract* eventType, the corresponding event specification written in the syntax above. In the figure, we shadow the submodel corresponding to the complex event `(SeqUnique(timeout Or eAck[IDc]) Before uploadSuccess)` to focus the attention of the reader on the most significant part of the event type specification w.r.t. the expressiveness of the proposed language. Indeed, to concisely and generally express such an event we use: *i)* event operators coming from both Drools [24] (i.e., the *Before* event operator) and GEM language [32] (i.e., the *Or* operator) showing the importance of the combination of the two reference event specification languages; and *ii)* the novel event operator *SeqUnique* we introduce to extend the expressiveness of the both reference languages. Without such combination and extension we would not have been able to express such complex event neither in Drools nor in GEM languages.

4. Using PMM models

The management of non-functional properties may involve all the phases of the MDE process. PMM can be used to support many different activities along the process. In this section, based on our experience within the CONNECT project, we focus on two key activities for the development of dynamic, evolving and heterogeneous systems that are monitoring and synthesis. Specifically, in Section 4.1 we present a model-driven configuration of a monitoring infrastructure by means of PMM models, whereas in Section 4.2 we show a synthesis approach taking into consideration performance concerns expressed by PMM models.

4.1. Monitoring of Non-Functional Properties

Event-based monitoring is a common approach for observing and analyzing the behavior of distributed systems [32, 28, 9]. To manually instruct the monitor about what raw data (events) to collect and how to infer whether or not a desired property is fulfilled may be an error-prone and time-consuming task. It would in fact require a substantial human effort and specialized expertise if the high-level intuitive description of the system properties to be observed have to be translated into specialised and lower-level monitor configuration directives. Moreover, this process would need to be iterated each time the properties to be monitored change. Hence, adopting a model-driven approach to monitor configuration is desirable.

We have implemented an approach to automatically convert the PMM properties, metrics and event specifications into a concrete monitoring setup [8, 33]. Precisely, as depicted in Figure 16, the editor provided along with PMM allows the software developer to specify a property or metric or event as a PMM-compliant model that represents a property to be monitored. This is then automatically translated by a Model2Code Transformer into a concrete monitoring configuration.

The advantage of adopting a model-driven approach is that it allows the monitor to use any complex event processing engine as long as a Model2Code Transformer transforms the property (metrics or event) model into the rule specification language of that processing engine. In our implementation, we took as reference the monitor infrastructure presented in [9] that natively includes the Drools Fusion complex event processor[3]

As an example of MDE monitor configuration, we show in Listing 1 the Drools code generated from the PMM models of the coverage property defined in Section 3.1.

Specifically, five rules are generated:

1. The first rule (line 1-21) counts the number of *eAlert_eAck* events that happen in a time window of 10 seconds and saves this information into a new generated event (called *counteAlert_eAck*). This rule refers to the *eAlert_eAck* event type showed in Figure 14. We remark that the *Seq* operator presented in this model is a new operator introduced in

[3]The Model2Code Transformer prototype from PMM to the Drools Fusion specification language, developed using Acceleo [1], is not presented here because it is not relevant for the paper purposes; we refer to [41].

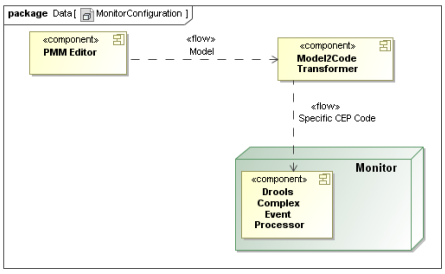


Figure 16: Property-driven monitor configuration.

PMM that is not included in the Drools operators; then this operator has been translated by the Model2Code Transformer in a sequence of *after* (line 14) operators of Drools.

2. The second rule (24-42) is similar to the first one: it takes in input the *deviceRegistration* eventType model of Figure 13 and generates the *countdeviceRegistration* event containing the number of *deviceRegistration* events.
3. The third rule (lines 45-62) captures the *counteAlert_eAck* and *countdeviceRegistration* events and computes a precise coverage measure, its value is then returned in the new generated event (named *percentage*, see line 59).
4. Similarly, the fourth rule (65-83) captures the *percentage* events and computes the average coverage measure. Note that in the third and fourth rules to compute the coverage measure the operators (*AVG* and *DIVISION*) specified in the *coverageTemplateModel* model depicted in Figure 9 are used.
5. Finally, the last rule (86-95) taking in input the *coveragePropertyModel* showed in Figure 8 checks that the computed average coverage measure is greater than the specified value (70) in the coverage property model.

```
1 declare Total_eAlert_eAckcaptured
2  @total: int
3 end
4
5 rule "Number of eAlert_eAck incomingEvents"
6 no-loop
7 salience 999
8 dialect "java"
9 when
```

```
10  $totaleAlert_eAck: Number();
11   from accumulate(
12  $event_eAlert_eAck: emergencyAlert( this before
13  $event_seq_eAck:
14  eAck( this after eAck after eAck after eAck after eAck after
        eAck after eAck after eAck after eAck after eAck after
        eAck after eAck after eAck after eAck after eAck after
        eAck after eAck after eAck after eAck after eAck )
15  )over window:time(10s) from entry-point "DEFAULT", count(
        $event_eAlert_eAck))
16 then
17  Total_eAlert_eAckcaptured counteAlert_eAck = new
        Total_eAlert_eAckcaptured();
18  counteAlert_eAck.setTotal($totaleAlert_eAck)
19  insert(counteAlert_eAck);
20  System.out.println( "Number of Incoming events: " +
        $totaleAlert_eAck);
21 end
22
23
24 declare Total_deviceRegistrationcaptured
25  @total: int
26 end
27
28 rule "Number of deviceRegistration incomingEvents"
29 no-loop
30 salience 999
31 dialect "java"
32 when
33  $totaldeviceRegistration: Number();
34   from accumulate(
35  $event_deviceRegistration: deviceRegistration
36  from entry-point "DEFAULT", count($event_deviceRegistration))
37 then
38  Total_deviceRegistrationcaptured countdeviceRegistration = new
        Total_deviceRegistrationcaptured();
39  countdeviceRegistration.setTotal($totaldeviceRegistration)
40  insert(countdeviceRegistration);
41  System.out.println( "Number of Incoming events: " +
        $totaldeviceRegistration);
42 end
43
44
45 declare CoveragePercentage
46  @Percentage: float
```

```
47 end
48
49 rule "Incoming coveragePercentage"
50  no-loop
51  salience 999
52  dialect "java"
53 when
54  $value: Float();
55   $eA: Total_eAlert_eAckcaptured() and  $dR:
         Total_deviceRegistrationcaptured();
56   from entry-point "DEFAULT"
57 then
58   value = Math.round( eA.total / dR.total )
59   CoveragePercentage percentage = new coveragePercentage();
60   percentage.setPercentage($value)
61   insert(percentage);
62 end
63
64
65 declare AVGCoveragePercentage
66  @Percentage: float
67 end
68
69 rule "AVGcoveragePercentage"
70  no-loop
71  salience 999
72  dialect "java"
73 when
74  $cp : CoveragePercentage()
75   $avg : Float()
76    from accumulate( CoveragePercentageItem ($value : value )
77    from entry-point "DEFAULT"
78    $avg : average( $value) )
79 then
80   AVGCoveragePercentage avgpercentage = new
         AVGCoveragePercentage();
81   avgpercentage.setPercentage($avg)
82   insert(avgpercentage);
83 end
84
85
86 rule "checkSatisfiedProperty"
87  no-loop
88  salience 999
89  dialect "java"
```

```
90 when
91   $cov : AVGCoveragePercentage( Percentage > 70 )
92   from entry-point "DEFAULT"
93 then
94   System.out.println( "Satisfied Coverage Property");
95 end
```

Listing 1: Drools Code generated by Model2Code Transformer for the *coverage* property

4.2. Synthesis of Non-Functional Properties

Today's networked environment is characterized by a wide variety of heterogeneous systems that dynamically interoperate to achieve some goal. In this evolving context, a-priori knowledge of the systems cannot be assumed, and automated solutions, such as mediator synthesis, appear to be the only way to achieve interoperability with the needed level of flexibility.

Figure 17 gives an overview of the context we consider. A number of heterogeneous networked systems, e.g., Tablet, Server, Smartphone and Desktop are dynamically available: heterogeneity may span various aspects and we focus on application layer heterogeneity. For instance, Smartphone in Figure 17 has a *shopping client* application represented by the shopping cart icon. Its shopping client is compatible/complementary with the Server *shopping server* application represented by the seller icon. However, due to some protocol discrepancies and non-functional concerns, they cannot seamlessly interoperate and a mediator that resolves their differences is needed in-between.

We consider that the networked systems to be mediated are black-box and describe in their interface: their interaction behavior, their owned non-functional properties, and possibly their non-functional requirements on the interactions with others systems. Non-functional concerns arise for the synthesized mediator, because in general a system that wants to achieve a goal

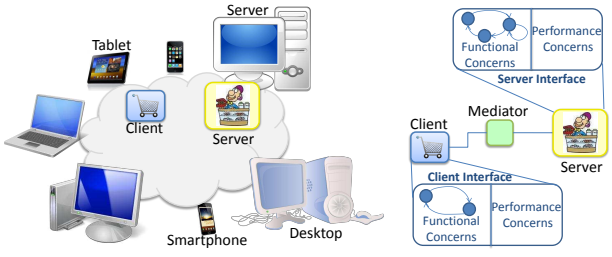


Figure 17: An overview of the Networked Systems context.

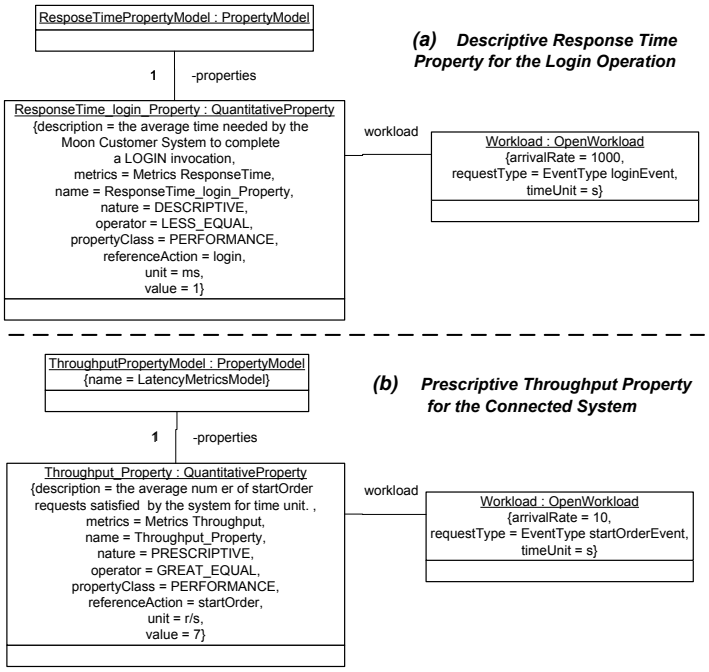


Figure 18: PMM models for Descriptive and Prescriptive Properties.

through the interaction with other systems, will yield some non-functional constraints, e.g. performance limits. It can happen that the synthesized mediator satisfies the functional concerns but, when assembled with the interacting systems, the connected system overall does not satisfy the performance requirements. Hence when synthesizing the mediator we need to consider the non-functional requirements along with the functional ones. Hence, we propose an approach to automated mediator synthesis [21], that takes into account performance concerns during the synthesis. To so this, we enhance the mediator synthesis with a performance analysis-based reasoning that acts on a preliminary intermediary mediator (produced before).

To specify non-functional concerns, i.e., the non-functional requirements the connected system must satisfy, as well as the non-functional characteristics of the individual systems and of the synthesized mediator actions, we use PMM. For the analysis we use the Æmilia Architectural Description Language (ADL) [7], based on the stochastic process algebra $EMPA_{gr}$ [6], which provides a formal architectural description of complex software systems allowing the performance analysis of the specified system.

In the performance analysis-based reasoning the PMM properties are used for two aims:

- the DESCRIPTIVE properties, specifying the performance characterization of the networked systems actions, are used to set the action rates in the Æmilia specification. In particular, in the generation of the Æmilia specification, an auxiliary action is generated when the descriptive response time or service rate property for a communication action is provided. For example, Figure 18.a shows the response time property (1 ms) of the action *login*. From this information we are able to set the auxiliary action rate as the inverse of the response time.

- the PRESCRIPTIVE properties, defining the performance requirements on the final system, are automatically translated into the required *measures specifications* that define the performance figures to be calculated during the performance analysis of the final system. For example, Figure 18(b) shows a PRESCRIPTIVE property representing a throughput requirement on a connected system.

More details of this enhanced synthesis approach and the implemented framework are given in [21].

5. Systematic Survey of Non-Functional Meta-Modeling

In order to compare PMM with existing similar approaches we performed a systematic survey on meta-models addressing non-functional properties, metrics and complex events languages. Below we briefly present first the research method and then the obtained results.

5.1. Research method

This survey has been conducted following the guidelines for systematic review in software engineering research proposed by Kitchenham [31]. These guidelines cover three aspects of a systematic review: planning the review, conducting the review and reporting results. In the planning phase we identified the goal of this review that is *to understand the current state of art in modeling non-functional properties and comparing the existing approaches.* Specifically, we formulated the following research questions (RQ):

RQ1: what approaches have been proposed for modeling non-functional properties?

RQ2: what type of non-functional property is addressed and in what application domain by the different modeling solutions?

RQ3: how are such approaches characterized in terms of expressiveness, formalization, purpose and automation?

RQ4: and finally, how similar/dissimilar are existing approaches to PMM?

We performed an automated search for retrieving the more relevant papers dealing with non-functional properties specification. Similarly to the review presented in [2] concerning MDD approaches dealing with non-functional properties, we launched our search on Web of Science (WoS). Specifically, we searched by topic (in title, abstract and keywords) using Science Citation Index Expanded (SCI-EXPANDED) and Conference Proceedings Citation Index- Science (CPCI-S) citation databases, and selected "English papers" from 1990 to 2012 (26 July).

According to the research goal, we defined the following search string:

```
(‘‘model driven’’ OR ‘‘meta-model’’ OR ‘‘meta model’’ OR
     ‘‘metamodel’’ OR ‘‘model-driven’’ OR ‘‘MDE’’)
                          AND
((‘‘non functional’’ OR ‘‘security’’ OR ‘‘dependability’’ OR
‘‘performance’’ OR ‘‘nonfunctional’’ OR ‘‘non-functional’’ OR
‘‘quality’’ OR ‘‘NFR’’) AND (‘‘requirements’’ OR ‘‘concepts’’
                   OR ‘‘properties’’))
                          OR
                     ‘‘metric*’’
                          OR
                     ‘‘events’’)
```

Specifically, our search string consists of four parts: a first part addressing meta-modeling proposals in model-driven systems, the second part is related to non-functional properties or requirements and in particular to security, dependability and performance ones, the last two parts target works that describe model-driven approaches dealing with metrics or events respectively. With respect to the query used in [2], which is the most similar survey to ours, we introduced in the search several new keywords that are specific to our research domain, such as meta-model, metric, event, security, dependability,

and performance. Since the new keywords are merged by OR operator, as a matter of fact we have widened the retrieved sample.

From this automatic search we obtained 853 papers, reduced to 74 after reading the title and the source, then reduced to 28 after reading the abstract and finally reduced to 25 after reading the full text.

For complementing the results of the automatic search, we included a few other relevant works [32, 18, 39, 27, 24, 38, 29, 40, 58, 17] that we had found by manual search but were not retrieved by the WoS-based search (mainly these refer to venues not indexed in WoS).

5.2. Results

Table 2 shows the classification framework we adopted in this study. The extracted data are classified according to seven dimensions that are: *type of non-functional property, domain, instrument, expressiveness, formalization, purpose and automation.* These are reported in the table columns. Specifically, the second and third columns answer RQ2, whereas columns from fifth to eighth answer RQ3. We detail each of them in the following:

Type of non-functional property or metric (NFP). It refers to the type of non-functional property (metric) that is the target of the modeling activity. The analyzed papers focus on one or more types of non-functional properties. Most of them address security, other ones performance or dependability. Other works are more generic and do not focus on a specific non-functional property.

Domain. It represents the technological context in which the non-functional properties are defined and evaluated. Modeling of non-functional properties fits in various application domains. Many approaches focus on Service-Oriented Architecture (SOA) applications and distributed systems, whereas other proposals address database management systems, data warehouses, safety-related standards, multi-agent systems and attack modeling. Different works deal with the specification of software metrics applied to software products such as system models, programs or requirements.

Instrument. It refers to the formalism used for expressing the non-functional properties. Several authors propose UML extensions (UML profile), others design an owned meta-model specifically conceived for expressing non-functional properties.

Expressiveness. With this concept we mean the power of the modeling solution in terms of what it is able to express, specifically properties (requirements, quality attributes) and/or metrics and/or events. Most approaches define properties (quantitative or quantitative), few works target metrics and events.

Formalization. It represents the level of formalization used by the different modeling languages for specifying a non-functional property. This dimension can be characterized as informal, which means described as a natural language piece of text, or in a semi-formal language using for instance names and values tied by a relational operator, or by a more formal description usually represented by a mathematical formula.

Purpose. The non-functional property models can be aimed to support different activities of the MDE process such as analysis, synthesis, deployment and monitoring. Most of the analyzed approaches have been defined for analysis and design purposes, few for monitoring or SLA specification, the other ones have specific purposes such as attack modeling, certification, software measuring.

Automation. It refers to the support provided by the different solutions in terms of automated facilities for properties specification and tools for the Model2Code (M2C) transformations. Some authors proposing an own meta-model provide an editor that represents a useful facility for models specification. In case of UML profile, the authors usually do not provide this facility since a UML profile can be handled by UML modeling tools.

Table 2: Classification of approaches dealing with modeling of non functional properties, metrics and events according to our systematic survey

Ref	**Type of NFP**	**Domain**	**Instrument**	**Expressiveness**	**Formalization**	**Purpose**	**Automation**
PMM	performance dependability security	any	own meta-model	properties metrics events	formal	synthesis monitoring	editor M2C transformer
Systematic Survey Results							
[5, 4]	security	distributed systems	UML's OCL formula	properties	formal	analysis	editor,

						design	analysis and validation tool
[42]	security	any	own meta-model	requirements	semi-formal	analysis design implementation	editor M2C transformer M2M transformer
[56]	security	active database	own meta-model	events	formal	monitoring	none
[57]	security	airborne systems based on RTCA DO-178B standard	UML profile	properties	informal	assessment certification	none
[11]	security	multi-agent systems	own meta-model	requirements	semi-formal	analysis design	none
[34]	security	distributed systems	own meta-model	requirements metrics	informal	attack modeling	none
[51]	security	data warehouses	UML profile	requirements	semi-formal	design	M2C transformer M2C transformer
[25]	any	information systems	own meta-model	quality attributes metrics	informal	design	none
[43]	security	business process	UML activity diagram profile	requirements	informal	analysis design	M2M transformer
[47, 37]	security	SOA business process	UML profile	requirements	informal	design	none
[46]	any	software products process models	own meta-model	metrics	formal	design specification model transformations	none
[54]	dependability security	socio-technical systems (business process)	own meta-model	requirements	semi-formal	analysis design	editor

[32]	any	distributed systems	own meta-model	events	formal	monitoring	none
[20]	any	distributed systems	own meta-model	properties	semi-formal	QoS models exchange	model repository and generic infrastructure to manage NFP
[55]	any	any	-	properties	informal	-	-
[16, 17]	any	service oriented systems	own meta-model	properties (SLA)	semi-formal	SLA specification monitoring	none
[49]	qualitative properties	any	UML profile	properties as softgoal	informal	softgoal representation satisfiability evaluation	none
[24]	any	any	own meta-model	events	formal	monitoring	none
[18]	any	any	own meta-model	events	formal	any	none
[39, 38] [29, 40]	performance safety	concurrent systems Real-Time Embedded System	UML profile	-	semi-formal	design, analysis simulation	code generation
[26]	performance reliability security	service oriented systems	UML profile	properties	semi-formal	analysis deployment	M2C transformer M2M transformer
[27]	dependability resilience	safety critical systems	own meta-model	properties	formal	design	none
[53]	security fault tolerance	SOA	UML profile	properties	semi-formal	specification of service oriented application concepts	M2C transformer
[35]	any	software models	own meta-model	metrics	formal	measuring software (programs, requirements,	editor measurement generator

						system architectures)	
[30]	Performance Reliability Availability	Service-oriented systems	UML profile	quality attributes requirements, metrics	formal	-	-
[13]	-	internal software	own meta-model	metrics	semi-formal ontology-based	web engineering	none
[12]							
[58, 44]							

Looking at the results summarized in Table 2, we can now also answer RQ4 and conclude that although valuable approaches exist for modeling non-functional properties, none of them proposes as a flexible and comprehensive solution as that of PMM. More precisely, the main advantages of PMM with respect to similar approaches are:

- it is a comprehensive meta-model addressing non-functional properties spanning over performance, dependability and security. The analyzed approaches either address only a subset of the properties addressed in PMM or they target general non-functional properties without specifying their type.

- it is generic and independent from the specific application domain. We applied it for monitoring and synthesis in different contexts, e.g. heterogenous dynamically connected systems [8], service choreography [3]. Most approaches are defined for a specific application domain and few ones are domain-independent as PMM.

- it is more expressive than all analyzed solutions since it represents the only proposal that allows for expressing properties, metrics and events. All the other approaches are centered in only one or two of these modeling activities.

- it is fully-fledged and can provide automated support in many phases of the MDE process. Specifically, it supports automated procedures for Model2Code and Model2Text transformation during the monitoring and synthesis activities respectively (as we showed in Section 4.1 and Section 4.2). PMM has an associated editor that allows for deriving

new properties, metrics and events, it is user-friendly and generated as an Eclipse plugin.

- it adopts a formal description for defining metrics by introducing the `MetricsTemplate` concept containing the mathematical definition of `Metrics` (see Section 2.2). Many proposed meta-models and UML profiles adopt a semi-formal or informal description.

Notwithstanding our efforts of being systematic and accurate, the survey is inevitably subject to threats to validity. The main threats we identify may undermine the search sample and the drawing of conclusions. Concerning the search samples, we could have missed some relevant solution, for example because this is not available in WoS. Although we made some manual search in other repositories, like the ACM Digital Library, and indeed we found other papers that have been added, still we might have left out other works that invalidate our conclusion about the uniqueness of PMM. Our measure to address this risk is the peer-to-peer review and subsequently the readers screening, who would be welcome to signal us these missing works. Concerning the validity of the conclusions, the comparison between our work and the survey findings is prone to our subjective opinion. We have explicitly reported our classifications in Table 2 so that they can be challenged by readers.

5.3. Most relevant approaches

While in Table 2 we sum up all the results of the systematic survey, in this section we focus on some most relevant works addressing modeling of non-functional properties, metrics and events, and we go deeper in discussing their similarities and differences with respect to PMM. We distinguish the approaches concerning modeling of non-functional properties and metrics from those defining events specification languages.

Property and Metrics Modeling.. Concerning the definition of properties and metrics the most similar approaches to PMM are [35, 26, 45].

In [35], which is an extended version of [36], Monperrus and coauthors propose a metric specification meta-model that represents an abstract and declarative specification of software metrics and includes all elements, required for the definition of metrics, that are independent of a specific domain. Thus the main concept behind PMM of specifying a metric as an instance of a metric specification meta-model is common to this work [35]. The approach

in [35] allows modelers to automatically add measurement capabilities to a domain specific modeling language. The target of the model driven measurement approach are software models (such as software architecture models, requirements models, real-time models or implementation models) of a domain specific modeling language. Differently from [35], the metrics defined using PMM are applied to software systems and services and allow to define temporal relations on the system entities (and not on models). Taking inspiration from Monperrus et al.'s work, PMM separates the property definition from the application domain. However, differently from [35], the PMM meta-model addresses specifically *non-functional property and metrics*. Moreover, it introduces additional concepts concerning the qualitative and quantitative properties definition and the events modeling. As PMM, the meta-model proposed in [35] has a formal description, it allows for defining arithmetic-based metrics and complex metric formula. An additional advantage of PMM with respect to the meta-model provided in [35] is the distinction between a generic metric formula (represented by a MetricsTemplate) and the concrete metric (i.e., the Metrics concept) instantiated by means of the EventSet concept to a specific application domain, represented by the domain of the software system the PMM is used for. This makes our meta-model more flexible and allows to reuse the same generic template in different scenarios. Finally, the metrics specified in [35] are applied to software models derived from the program execution, and as the same authors claim, these metrics cannot be applied for computing time constraints of the execution context. The authors plan to improve their approach for taking into account also runtime models. On the contrary, the metrics specified by means of PMM can be dynamically instantiated for deriving runtime measures of the system behavior. This is obtained by dynamically instructing a runtime monitoring system with the metrics to be computed following the model driven approach to monitor configuration presented in Section 4.1.

In [26] Gilmore and coauthors propose a thorough approach to model-driven development of service-oriented systems that consider non functional properties. They propose an extension of the UML4SOA Profile, called UML4SOA-NFP, to take into account non-functional properties. UML4SOA-NFP is used together with the UML Marte Profile to represent performance, security and dependability properties for service-oriented systems. The UML models enriched with the non-functional properties are used for formal analysis of performance and dependability properties based on PEPA stochastically timed process algebra and to decide the deployment of the system. This

last step is supported by Model2Model and Model2Text transformations implemented in VIATRA [52]. In UML4SOA-NFP Profile, the non-functional properties are part of the contract signed between two participants, a requester and a provider, and agreed for a service interface. Each property has a *NFcharacteristic* (similar to our `propertyClass`) representing the non-functional aspects the property refers to (such as performance, security, etc...) and each *NFCharacteristic* can contain one or more *NFDimension* represented as an attribute name and a set of values. The attribute name represents the specific metrics to be considered (such as, throughput or response time for performance *NFCharacteristic*) and the value(s) is(are) the one(s) that must be guaranteed at runtime. The modeling and the idea behind the proposed process in [26] yields many similarities to our approach but also several differences. First, [26] is suited to SOA applications while PMM has been devised to be independent from a specific application domain and development paradigm. Hence in PMM there are not concepts such as participant and contract, whereas the non functional properties are classified as `DESCRIPTIVE` (an owned property) or `PRESCRIPTIVE` (representing a requirement). UML4SOA-NFP does not provide a formal specification of the metrics (*NFDimension*) the property refers to. The semantics of the metrics is implicit in the name of the dimension (e.g., *averageRespTime* or *maxRespTime*). In PMM instead we introduce the concepts of `Metrics` and `MetricsTemplate` to fill this gap. Indeed, the PMM `Metrics` concept is similar to the UML4SOA-NFP *NFDimension*. The difference is that it refers to a `MetricsTemplate` that defines a mathematical function to be used to calculate the value of the metrics itself, providing the semantics missing in UML4SOA-NFP. Moreover, PMM contains the concepts of `EventType` and `EventSet` that are used by the system monitoring to verify at runtime the validation of the specified non functional property. While PMM is a stand-alone meta-model implying that it can be embedded in several software modeling languages by spending an effort to integrate it in the specific development language, UML4SOA-NFP is suited for UML implying that it can be straightforwardly used in UML based software life-cycle, but on the other side its use in other modeling languages requires some effort. For what concerns the usage of the models conforming to the two meta-models, UML4SOA-NFP allows the integration with the performance and reliability analysis and with the deployment mechanisms by means of Model2Model and Model2Text transformations; PMM models can be used in several directions, for runtime synthesis of mediators and dynamic configuration of

runtime monitoring as we showed in Section 4.

Finally, [45] has several similarities with our approach. We share with [45] the ideas of property, generic metrics and application-domain metrics definition, of a specific language to specify them and of the usage of transformation to translate the models in different domains. However, PMM differs from this work for several reasons: *i)* PMM does not target specific paradigm, whereas [45] specifically targets component-based systems; *ii)* in PMM the semantics of a metrics is explicitly demanded to the users which must specify the event(s) associated to the metrics, in [45] instead a metrics has associated a fixed set of semantics; *iii)* PMM deals with quantitative and qualitative properties (these last are non-functional properties that have not associate a metrics such as security), [45] instead allows the modeling of quantitative properties.

Events Modeling.. Defining expressive complex event specification languages has been an active research topic for years [32, 56, 18, 24]. The most similar approaches to PMM complex events specification language are GEM [32] and Drools Fusion [24].

GEM [32] is a declarative and interpreted events language. It is rule-based (similarly to other event-condition-action approaches) and has been designed for monitoring of distributed systems. It allows for specifying abstract events as combinations of low-level events coming from different nodes and for defining various temporal constraints for events compositions. Differently from GEM, the PMM events language is not specifically conceived for monitoring purposes, in fact it can be also used for synthesis activities (see Section 4.2) and along other phases of the MDE process. Similarly to GEM, our proposed language allows for specifying primitive and arbitrarily composite events. Both approach consider an event as denoting that something of interest happened, but whereas in GEM an event occurs instantaneously, in PPM more accurate time constraints can be specified. Specifically, in PMM a simple event has a start and end timestamp and for complex events it is possible to specify some parameters for quantifying the maximum and minimum temporal distance between the time when the correlated event finishes and the current event starts. In GEM, five event composition operators are specified: *and*, *after*, *interleaving*, *or* and *before*. They can be used to define other ones. As presented in Section 2.3, PMM provides a more powerful and complete set of compositions operators including those of GEM.

Drools Fusion [24] is a complex event processing engine that supports

among the other goals a language for the complex events specification. In Drools Fusion the events are special entities that represent an important change of state in the application domain and have strong temporal constraints and relationships. Every event has an associated *timestamp* and *duration*. The Drools open-source event processing engine performs composition and aggregation of events. This engine can be fully embedded in existing Java architectures and provides efficient rule processing mechanisms. The set of composition operators used in Drools Fusion supports temporal parameters and sliding windows of interesting events in order to model the temporal relationships between events. Specifically, Drools implements 13 temporal operators: *after*, *before*, *coincides*, *during*, *finishes*, *finished by*, *includes*, *meets*, *met by*, *overlaps*, *overlapped by*, *starts*, *started by*. As presented in Section 2.3, PMM complex events language includes all temporal operators of Drools Fusion and improves their definition distinguishing operators with the same type but different number of parameters and defining the order in which the events appear in the composition.

As said, the complex events language embedded into PMM combines features of GEM and Drools Fusion languages and in addition presents new operators not included in the considered languages. Specifically, from one side, GEM has few basic composition operators that are used for defining the others but they are not enough for expressing all compositions operators of an event-based monitor [9] that includes a Drools Fusion engine. From the other side, Drools Fusion does not give the possibility to express operators such that *or*, *not* or *seq*. The PMM events language (presented in Section 2.3) combines the advantages of both languages and provides the developer with a more powerful, flexible and expressive specification language. As showed in Section 4.1, an additional advantage of PMM events specification language is to be machine-processable and then it can be easily translated into rules of Drools Fusion engine. We refer to [10] for a detailed comparison of PMM complex events operators and those of GEM and Drools Fusion.

Composite event specification and detection has received much attention also in the active database area [14, 56]. [56] proposes an event-condition-action approach especially developed for active databases, supporting temporal and composite events specification. The authors introduce a formal meta-model for defining the events, taking into account the semantics of the events presented into Snoop [14] that is another event specification language for active databases. The meta-model proposed in [56], as PMM, distinguishes primitive from complex event and is based on tree dimensions: the

event type pattern describing the overall structure of the complex event; the event instance selection defining what instances need to be bound to a complex event; and the event instance consumption that determines the invalid instance that cannot be considered for the detection of further complex events. The event type pattern and the event instance selection concepts are similarly expressed in PMM whereas the event instance consumption concept is not defined in PMM since it is related to the runtime detection activity of events. Using PMM for property-driven monitoring configuration, this detection activity is demanded to the monitoring system, then an event instance is consumed when it is observed and collected by the complex events processor embedded in the monitoring infrastructure. Concerning the events composition operators, the meta-model presented in [56] provides a basic set of operators composed by: *sequence*, *simultaneous*, *conjunction*, *disjunction* and *negation*. PMM allows to express all these operators and provides other operators enhancing the completeness and expressiveness of the modeling language.

Other proposals focus on formally defined approaches. Among them, TESLA [18] has a simple syntax and a semantics based on a first order temporal logic. The authors of TESLA [18] also show as TESLA rules can be interpreted by a processing system, having an efficient event detection algorithm based on automata. TESLA considers incoming data items as notifications of events and defines how complex events can be defined from simpler ones. For specifying events, it provides: content and temporal constraints, parameterizations, negations, sequences, aggregates. All these concepts can be expressed also by PMM. Specifically, as in TESLA, in PMM events selection is based on temporal condition (for instance all event occurrences specified into a `timeWindow` of an `eventSet` are modeled) or parameters values. The negation and sequence concepts of TESLA are expressed in PMM by means of the `Not` and `Seq` complex events operators whereas for expressing the aggregates PMM uses the mathematical operators of the *MetricsTemplate* model. An interesting aspect of TESLA is that it provides the ability of specifying fully customizable policies for event selection and consumption. This feature allows for expressing the events iteration without explicitly using the Kleene operator. In PMM only bounded iterations can be specified using the `Seq` operator and the associated `minLenght` parameter. However, the clear and easy-to-use syntax of TESLA allows for specifying a limited number of different operators. Specifically, TESLA provides three event composition operators: *each-within*, *first-within*, and *last-within*. With respect

to TESLA, our work provides a more high-level and more specialized complex events specification language included into a comprehensive and flexible meta-model. Finally, [19] provides an extensive survey of information flow processing systems and gives an overview of the languages models adopted by those systems, listing their type and the set of available operators.

peculiarities of these systems. implemented. The paper also highlights that, even if several modeling languages have bee defined in literature, the MDD does not embed them in the design models and hence in the whole software life-cysle, making the software models and the NFP specification two separate islands. consider property composition, and differences between evaluated, basic or constrained properties. Finally, the paper presents a generic framework for QoS management that, using MOF capabilities, allows to automatically generate a QoS model repository and XMI documents to exchange models in a notation independent from the QoS specific notation. In our approach instead we use MDE techniques to bridge the NFP models specification towards NF models analysis, NFP monitoring, testing and software synthesis taking into account also NFP. in several domains. Moreover, PMM allows the definition of new metrics the properties to be validated refer to, and events that link the properties to observable evidences of the system behavior execution. Instead, both the approaches define a process that is model-based and joins the specification of NFPs with their validation.

6. Conclusion

We have presented PMM, a generic, comprehensive and flexible meta-model for the modeling of non-functional properties, metrics and complex events. The solution here presented and made available to the community is the result from our research work in the CONNECT project. The meta-model has been progressively augmented and refined all along the four years of the project duration. The work was triggered by the need to express non-functional requirements in the dynamic composition of heterogeneous networked systems. As no single existing approach was sufficient to our purposes, our intent was to coalesce into one single source the best aspects of several existing approaches, and to unify the instruments to model both properties of interest and complex events. In fact, it is through the observation of complex events that we can assess whether non-banal non-functional properties are verified.

We have included in the paper a few examples of models instantiated from PMM, as well as two applications of PMM in the synthesis and monitoring. Other examples of property definitions can be found on the PMM page [41]. We intend to continue experimentation in different scenarios and context; Other applications are planned. In fact, PMM has been conceived with no specific purpose in mind, and can be used all along the lifecycle of dynamic heterogenous systems, including requirement analysis and testing. Currently, we are already employing PMM for driving analysis and monitoring of choreographies [3].

We briefly sketched in [33] the issues related to the usage of PMM during the testing phase. Specifically, both prescriptive, and descriptive properties can drive the test-case generation task. Prescriptive properties can be used as contracts (e.g. SLA, etc) that a system under test should abide by while descriptive properties can regulate integration testing process. Also, the outcome of the property modeling framework contributes in the definition of the expected behavior of the test case execution. We want to investigate more in this direction providing experimental results.

To assess PMM, we performed a systematic survey of related work. The comparison shows the main advantages of PMM with respect to existing solutions: PMM represents the only proposal that allows for expressing properties, metrics and events; it is independent from the specific application domain; it addresses different kinds of non-functional properties, similar solutions target only a subset of the properties addressed in PMM; it provides automated support for most phases of the MDE process, specifically for monitoring and synthesis.

Acknowledgments

This work has been partially supported by the European Project CONNECT Grant Agreement No.231167.

References

[1] Acceleo, `http://www.eclipse.org/acceleo/`.

[2] David Ameller, Xavier Franch, and Jordi Cabot, *Dealing with non-functional requirements in model-driven development*, 18th IEEE International Requirements Engineering Conference (RE), 2010, pp. 189 –198.

[3] Cesare Bartolini, Antonia Bertolino, Andrea Ciancone, Guglielmo De Angelis, and Raffaela Mirandola, *Quality requirements for service choreographies*, WEBIST 2012 - Proceedings of the 8th International Conference on Web Information Systems and Technologies, SciTePress, 18 - 21 April, 2012, pp. 143–148.

[4] David Basin, Manuel Clavel, Jürgen Doser, and Marina Egea, *Automated analysis of security-design models*, Inf. Softw. Technol. **51** (2009), no. 5, 815–831.

[5] David Basin, Manuel Clavel, Jrgen Doser, and Marina Egea, *A metamodel-based approach for analyzing security-design models*, Models, 2007, pp. 190–204.

[6] Marco Bernardo and Mario Bravetti, *Performance measure sensitive congruences for markovian process algebras*, Theor. Comput. Sci. **290** (2003), no. 1, 117–160.

[7] Marco Bernardo, Paolo Ciancarini, and Lorenzo Donatiello, *Architecting families of software systems with process algebras*, ACM TOSEM **11** (2002), 386–426.

[8] Antonia Bertolino, Antonello Calabrò, Francesca Lonetti, Antinisca Di Marco, and Antonino Sabetta, *Towards a model-driven infrastructure for runtime monitoring*, Proceedings of SERENE, 2011, pp. 130–144.

[9] Antonia Bertolino, Antonello Calabrò, Francesca Lonetti, and Antonino Sabetta, *Glimpse: a generic and flexible monitoring infrastructure*, Proceedings of EWDC, 2011, pp. 73–78.

[10] Antonia Bertolino, Antinisca Di Marco, and Francesca Lonetti, *Complex events specification for properties validation*, Proceedings of 8th International Conference on the Quality of Information and Communications Technology (QUATIC), 3-6 September 2012, Lisbon, Portugal, pp. 85–94.

[11] Ghassan Beydoun, Graham Low, Haralambos Mouratidis, and Brian Henderson-Sellers, *A security-aware metamodel for multi-agent systems (MAS)*, Information and Software Technology **51** (2009), no. 5, 832 – 845.

[12] Marko Bošković and Wilhelm Hasselbring, *Model driven performance measurement and assessment with modepemart*, Proc. of MODELS, 2009, pp. 62–76.

[13] Cristina Cachero, Coral Calero, and Geert Poels, *Metamodeling the quality of the web development process' intermediate artifacts*, Proceedings of the 7th international conference on Web engineering, ICWE'07, 2007, pp. 74–89.

[14] Sharma Chakravarthy and D. Mishra, *Snoop: An expressive event specification language for active databases*, Data & Knowledge Engineering **14** (1994), no. 1, 1–26.

[15] CONNECT Consortium, *Deliverable 6.1-Experiment scenarios, prototypes and report. http://connect-forever.eu/*, 2011.

[16] Anacleto Correia and Fernando Brito e Abreu, *Model-driven service level management*, 4th International Conference on Autonomous Infrastructure, Management and Security, AIMS 2010, Zurich, Switzerland, June 23-25, 2010. Proceedings, Lecture Notes in Computer Science, vol. 6155, Springer, pp. 85–88.

[17] Anacleto Correia, Fernando Brito e Abreu, and Vasco Amaral, *Slalom: a language for sla specification and monitoring*, CoRR **abs/1109.6740** (2011).

[18] Gianpaolo Cugola and Alessandro Margara, *TESLA: a formally defined event specification language*, Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems, DEBS '10, 2010, pp. 50–61.

[19] ———, *Processing flows of information: From data stream to complex event processing*, ACM Comput. Surv. **44** (2012), no. 3, 15:1–15:62.

[20] Jérôme Daniel, Bruno Traverson, and Sylvie Vignes, *A QoS Meta Model to Define a Generic Environment for QoS Management*, Trends in Distributed Systems: Towards a Universal Service Market, Lecture Notes in Computer Science, vol. 1890, 2000, pp. 334–339.

[21] Antinisca Di Marco, Paola Inverardi, and Romina Spalazzese, *Synthesizing connectors meeting functional and performance concerns*, FASE, Submitted for review, 2013.

[22] Antinisca Di Marco, Claudio Pompilio, Antonia Bertolino, Antonello Calabrò, Francesca Lonetti, and Antonino Sabetta, *Yet another meta-model to specify non-functional properties*, Proceedings of QASBA, 2011, pp. 9–16.

[23] Nicola Dragoni, Fabio Martinelli, Fabio Massacci, Paolo Mori, Christian Shaefer, Thomas Walter, and Eric Vetillard, *Security-by-contract (SxC) for software and services of mobile systems*, At your service - Service-Oriented Computing from an EU Perspective., MIT Press, 2008.

[24] Drools Fusion: Complex Event Processor, `http://www.jboss.org/drools/drools-fusion.html`.

[25] Auvo Finne, *Towards a quality meta-model for information systems*, Software Quality Control **19** (2011), no. 4, 663–688.

[26] Stephen Gilmore, László Gönczy, Nora Koch, Philip Mayer, Mirco Tribastone, and Dániel Varró, *Non-functional properties in the model-driven development of service-oriented systems*, Software & Systems Modeling **10** (2011), 287–311 (English).

[27] Nicolas Guelfi, *A formal framework for dependability and resilience from a software engineering perspective*, Central European Journal of Computer Science **1** (2011), 294–328 (English).

[28] Sam Guinea, Gabor Kecskemeti, Annapaola Marconi, and Branimir Wetzstein, *Multi-layered monitoring and adaptation*, Proceedings of the 9th international conference on Service-Oriented Computing, ICSOC'11, 2011, pp. 359–373.

[29] Muhammad Zohaib Iqbal, Andrea Arcuri, and Lionel Briand, *Code generation from uml/marte/ocl environment models to support automated system testing of real-time embedded software*, Tech. Report 2011-04, Version 2, Simula Research Laboratory, Technical Report (2011-04), Version 2, 2011.

[30] Ivan J. Jureta, Caroline Herssens, and Stéphane Faulkner, *A comprehensive quality model for service-oriented systems*, Software Quality Control J. **17** (2009), 65–98.

[31] Barbara Kitchenham, *Procedures for performing systematic reviews*, Joint Technical Report, Keele University Technical Report TR/SE-0401 and NICTA Technical Report 0400011T.1, 2004.

[32] Masoud Mansouri-Samani and Morris Sloman, *GEM: a generalized event monitoring language for distributed systems.*, Distributed Systems Engineering **4** (1997), no. 2, 96–108.

[33] Antinisca Di Marco, Francesca Lonetti, and Guglielmo De Angelis, *Property-driven software engineering approach*, Proceedings of Fifth International Conference on Software Testing, Verification and Validation, 2012, pp. 966–967.

[34] André Miede, Nedislav Nedyalkov, Christian Gottron, André Konig, Nicolas Repp, and Ralf Steinmetz, *A generic metamodel for it security attack modeling for distributed systems*, International Conference on Availability, Reliability, and Security, feb. 2010, pp. 430 –437.

[35] Martin Monperrus, Jean-Marc Jézéquel, Benoit Baudry, Joël Champeau, and Brigitte Hoeltzener, *Model-driven generative development of measurement software*, Software and System Modeling **10** (2011), no. 4, 537–552.

[36] Martin Monperrus, Jean-Marc Jézéquel, Joël Champeau, and Brigitte Hoeltzener, *A model-driven measurement approach*, Model Driven Engineering Languages and Systems, 11th International Conference, MoDELS 2008, Toulouse, France, September 28 - October 3, 2008. Proceedings, Lecture Notes in Computer Science, vol. 5301, 2008, pp. 505–519.

[37] Jafreezal Jaafar Muhammad Qaiser Saleem and Mohd Fadzil Hassan, *SECURE BUSINESS PROCESS MODELLING OF SOA APPLICATIONS USING UML-SOA-SEC*, International Journal of Innovative Computing, Information and Control **8**, 2729–2746 (English).

[38] Luis Gabriel Murillo, Marcello Mura, and Mauro Prevostini, *Semi-automated Hw/Sw Co-design for embedded systems: from MARTE mod-*

els to SystemC simulators, Forum on Specification Design Languages, 2009. FDL 2009., sept. 2009, pp. 1–6.

[39] OMG, *UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE)*, http://www.omg.org/omgmarte/Specification.htm/.

[40] Papyrus for MARTE, `http://www.papyrusuml.org/scripts/home/publigen/content/templates/show.asp?P=137&L=EN&SYNC=Y`.

[41] PMM: Property Meta Model, `http://labse.isti.cnr.it/tools/pmm`.

[42] Óscar Sánchez Ramón, Fernando Molina, Jesús García Molina, and José Ambrosio Toval Álvarez, *Modelsec: A generative architecture for model-driven security.*, J. UCS **15** (2009), no. 15, 2957–2980.

[43] Alfonso Rodríguez, Eduardo Fernández-Medina, Juan Trujillo, and Mario Piattini, *Secure business process model specification through a uml 2.0 activity diagram profile*, Decis. Support Syst. **51** (2011), no. 3, 446–465.

[44] Simone Röttger and Steffen Zschaler, *Tool support for refinement of non-functional specifications*, Software and System Modeling **6** (2007), no. 2, 185–204.

[45] Simone Röttger and Steffen Zschaler, *Tool Support for Refinement of Non-functional Specifications*, Software and Systems Modeling **6** (2007), no. 2, 185–204.

[46] Motoshi Saeki and Haruhiko Kaiya, *Measuring characteristics of models and model transformations using ontology and graph rewriting techniques*, Evaluation of Novel Approaches to Software Engineering, Communications in Computer and Information Science, vol. 69, 2010, pp. 3–16.

[47] Muhammad Qaiser Saleem, Jafreezal Jaafar, and Mohd. Fadzil Hassan, *Security modeling of soa system using security intent dsl.*, ICSECS (3), Communications in Computer and Information Science, vol. 181, Springer, 2011, pp. 176–190.

[48] Bran Selic, *From model-driven development to model-driven engineering, keynote talk*, Real-Time Systems, 2007. ECRTS '07. 19th Euromicro Conference on, July 2007, p. 3.

[49] Sam Supakkul and Lawrence Chung, *A uml profile for goal-oriented and use case-driven representation of nfrs and frs*, Software Engineering Research, Management and Applications, ACIS International Conference on **0** (2005), 112–121.

[50] The CONNECT Project, `http://www.connect-forever.eu`.

[51] Juan Trujillo, Emilio Soler, Eduardo Fernández-Medina, and Mario Piattini, *A uml 2.0 profile to define security requirements for data warehouses*, Comput. Stand. Interfaces **31** (2009), no. 5, 969–983.

[52] Dániel Varró and András Balogh, *The model transformation language of the viatra2 framework*, Sci. Comput. Program. **68** (2007), no. 3, 187–207.

[53] Hiroshi Wada, Junichi Suzuki, and Katsuya Oba, *A model-driven development framework for non-functional aspects in service oriented architecture*, Int. J. Web Service Res. **5** (2008), no. 4, 1–31.

[54] Nicola Zannone, *The si* modeling framework: metamodel and applications*, International Journal of Software Engineering and Knowledge Engineering **19** (2009), no. 5, 727–746.

[55] Liming Zhu and Yan Liu, *Model driven development with non-functional aspects*, Proceedings of the 2009 ICSE Workshop on Aspect-Oriented Requirements Engineering and Architecture Design (Washington, DC, USA), EA '09, IEEE Computer Society, 2009, pp. 49–54.

[56] Detlef Zimmer and Rainer Unland, *On the semantics of complex events in active database management systems*, Proceedings of the 15th International Conference on Data Engineering, ICDE '99, 1999, pp. 392–399.

[57] Gregory Zoughbi, Lionel Briand, and Yvan Labiche, *Modeling safety and airworthiness (rtca do-178b) information: conceptual model and uml profile*, Softw. Syst. Model. **10** (2011), no. 3, 337–367.

[58] Steffen Zschaler, *Formal specification of non-functional properties of component-based software systems*, Software and System Modeling **9** (2010), no. 2, 161–201.

An approach to adaptive dependability assessment in dynamic and evolving connected systems

Felicita Di Giandomenico, Antonia Bertolino, Antonello Calabrò, Nicola Nostro

Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo"
Consiglio Nazionale delle Ricerche
via Moruzzi 1, 56100 Pisa, Italy
(digiandomenico, antonia.bertolino, antonello.calabro, nicola.nostro)@isti.cnr.it

Abstract

Complexity, heterogeneity, interdependency and, especially, evolution of system/services specifications, related operating environments and user needs, are more and more highly relevant characteristics of modern and future software applications. Taking advantage of the experience gained in the context of the European project CONNECT, which addresses the challenging and ambitious topic of eternally functioning distributed and heterogeneous systems, in this paper we present a framework to analyse and assess dependability and performance properties in dynamic and evolving contexts. The goal is to develop an adaptive approach by coupling stochastic model-based analysis, performed at design time to support the definition and implementation of software products complying with their stated dependability and performance requirements, with run-time monitoring to re-calibrate and enhance the dependability and performance prediction along evolution. The proposed framework for adaptive assessment is described and illustrated through a case study. To simplify the description while making more concrete the approach under study, we adopted the setting and terminology of the CONNECT project.

Key words: Adaptation, Dependability, Evolving Heterogenous Systems, Model-based Assessment, Monitoring, Performance

1. Introduction

Modern software applications are increasingly pervasive, dynamic and heterogeneous. More and more they are conceived as dynamically adaptable and evolvable sets of components that must be able to modify their behaviour at run-time to

tackle the continuous changes happening in the unpredictable *open-world* settings [3]. Operating in the open-world poses a number of unprecedented challenges to software systems, including:

- The reference specification of expected/correct operation is not a-priori available;
- Specifications are learnt/inferred, thus they can be incomplete, unstable, uncertain, with impact on all the software engineering processes built upon system specification;
- System components are assembled dynamically, with potential strong impact on interoperability in presence of heterogeneity;
- Assessment activities must accommodate change (and must be adaptable themselves), therefore special emphasis is on run-time assessment (possibly coupled with off-line analysis techniques, wherever possible), which is a new paradigm with respect to traditional assessment methods.

As a result of such prominent trends two related needs emerge.

On the one side, we observe that the interconnected components, which we refer to as the Networked Systems (NSs), are independently developed. Because of this, the fast pace at which technology advances along diverging tracks can form gaps and establish separately evolving technological islands, between which communication is hampered. Thus the state of practice is that ad hoc bridging solutions need to be continuously developed to fill those communication gaps.

On the other side, the everyday life of modern and future society is growingly depending on the services provided by such highly complex and pervasive systems. In some cases their failures might even lead to catastrophic consequences in terms of damages to human life, environment, economy. Therefore, dependability and performance properties of such systems become increasingly critical.

The European FP7 Future and Emerging Technology Project CONNECT addresses both needs, aiming at enabling *seamless* and *dependable* interoperability among NSs in spite of technology diversity and evolution. The ambitious goal of the project is to have eternally functioning distributed systems within a dynamically evolving open-world context. This is pursued through the on-the-fly synthesis of the CONNECTors through which heterogeneous NSs can communicate in dependable and secure way. Indeed, effective interoperability requires

to ensure that such on-the-fly CONNECTed systems provide the required non-functional properties and continue to do so even in presence of evolution, thus calling for enhanced and adaptive assessment frameworks.

In the context of the CONNECT project, approaches to both off-line (pre-deployment) and run-time analysis are under development to analyse and ensure the synthesis of CONNECTors with required dependability and performance levels. In particular, an assessment framework is proposed which combines stochastic model-based analysis with continuous on-line assessment of non-functional properties through a lightweight flexible monitoring infrastructure. The goal is to assess complex dependability and performance metrics through accurate analysis that adapts to the evolving context. Although not novel in its basic principles, this off-line and run-time integrated framework is proposed as a general, automated approach to fulfill the dependability and performance assessment needs in dynamic and evolving contexts.

In this paper, we initially point out the challenges of assessing non functional properties in dynamic CONNECTed systems and provide the context for our research objectives (Section 2). Then we introduce first separately the pre-deployment analysis method (Section 3) and the run-time monitor (Section 4) under development and hence their synergic usage (Section 5), through which adaptive assessment is pursued. A case study is also included (Section 6) to demonstrate the applicability of the integrated analysis framework. Finally we overview related work (Section 7) and draw conclusions (Section 8).

2. Context

Before introducing our approach, in this section we set the reference context within which we settled our study on dependability and performance assessment methodologies able to account for and adapt to system and environment changes. In the following two sub-sections we first provide a brief overview of the already mentioned CONNECT project, tailored to investigate research on eternally connected systems despite heterogeneity and dynamic evolution, and then discuss some emerging issues when addressing the assessment of systems in such context.

2.1. *Overview of the EU* CONNECT *project*

Our research is carried out in the context of the FP7 "ICT forever yours" European Project CONNECT[1], belonging to the Future and Emerging Technologies track. CONNECT collects a consortium of partners whose expertise covers middleware, software engineering, formal methods, machine learning, software synthesis and systems dependability. The CONNECT world envisions dynamic environments populated by technologically heterogeneous Networked Systems (NSs), and by the components of the CONNECT enabling architecture, called the CONNECT enablers.

The ambition of the project is to have eternally functioning systems within a dynamically evolving context. To overcome interaction protocol heterogeneity at all layers, the project introduces a revolutionary approach that dynamically generates the inter-mediator components to connect heterogeneous systems. This is achieved by synthesising *on-the-fly* the CONNECTors through which the NSs communicate. The resulting emergent CONNECTors then compose and further adapt the interaction protocols run by the CONNECTed System. In brief, the NSs manifest the intention to connect to other NSs. The enablers are networked entities that incorporate all the intelligence and logic offered by CONNECT for enabling the required connection. The emergent connectors produced by the action of enablers are called the CONNECTors, whereas as an outcome of the successful creation and deployment of CONNECTors we obtain the CONNECTed systems.

In Figure 1 we provide an overview of the CONNECT vision and architecture. We show in schematic form the enablers which are currently part of the CONNECT enabling architecture. From top to bottom, we see:

Discovery Enabler catches the requests for communication coming from the NSs and initiates the CONNECT process. We tend to make the minimum possible assumption on the information (called the *affordance*) that NSs must provide;

Learning Enabler : we use active learning algorithms to dynamically determine the interaction behaviour of a NS and produces a model in the form of a labeled transition system (LTS);

Synthesis Enabler : from the models of the two NSs, this enabler synthesises a mediator component through automated behavioural matching;

[1] http://connect-forever.eu

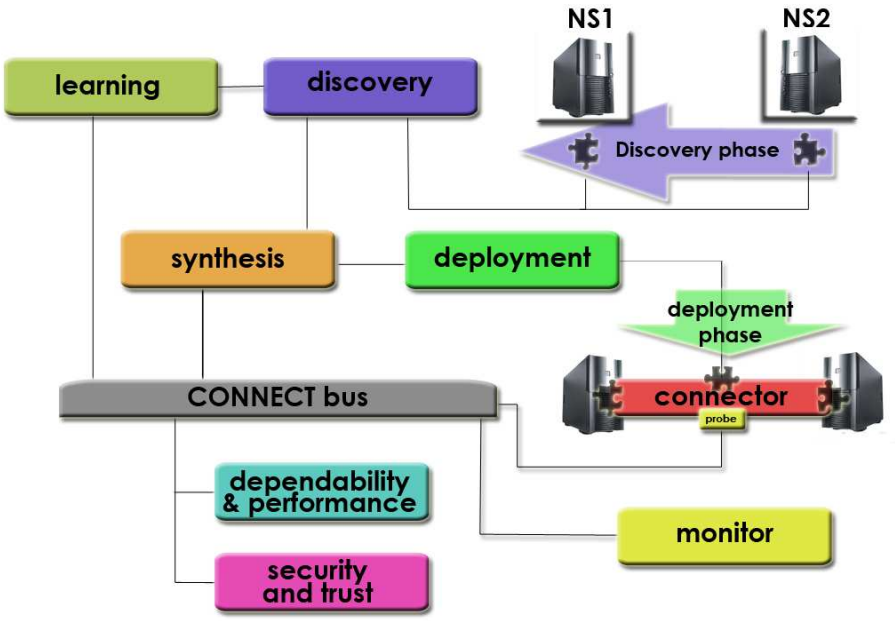


Figure 1: The CONNECT Architecture

Deployment Enabler finally deploys and manages the CONNECTors.

Evidently, such a dynamic context strongly relies on one side on mechanisms for ensuring dependability, security and trust, and on the other side on functional and non-functional behaviour monitoring, through which run-time adaptation of CONNECTors is triggered. Hence the CONNECT architecture also includes the following important enablers:

Monitoring Enabler collects raw information about the CONNECTors behaviour and passes them to the enablers (the monitor's customers) who requested them. The CONNECT monitoring infrastructure is further described in Section 4;

CONNECT bus : all communication among the enablers and with the CONNECTors happens through a message bus, which is currently implemented by a simple message-based communication model as for instance the Java Messaging Service (JMS);

DEPER Enabler : this is the main focus of this paper and is described in detail in the next section;

Security and Trust Enabler collaborates with the synthesis enabler to satisfy possible security and trust requirements. It also continuously determines if the requirements are maintained at run-time, by receiving monitoring data from the monitoring enabler (similarly to the integrated approach we exemplify for dependability and performance in Section 5).

2.2. Challenges in dependability and performance assessment in evolving context

The need for research advancement in the assessment of evolving, ubiquitous systems is recognized by the dependability/resilience community, being indicated as one of the prominent research challenges in the research agenda set up by the ReSIST European Network of Excellence [10]. In fact, it is observed that, since current and future systems result from evolutions of pre-existing systems, as a consequence assessment should move from off-line and pre-deployment, to continuous and automated operational assessment. The traditional approaches to assessment, which dominate the current practice, are: i) pre-deployment assessment, i.e. collecting data in a simulated environment (e.g. "model-based analysis", "statistical testing", "dependability benchmarking", etc.), and/or ii) processing the measurement data accumulated in real operation at a later stage, e.g. periodic reviews widely used in some safety-critical industries such as the nuclear sector. Both these categories of methods have shortcomings when dealing with evolution and dynamicity of the system under analysis. In fact, dealing with evolution and dynamicity raises two major challenges from the point of view of dependability and performance analysis:

- Pre-deployment assessment is limited by its nature: the impact on system dependability/performance cannot be known for unforeseen environments. Therefore, given the many possible variations occurring during software application lifetime, it would be necessary to analyse beforehand, through off-line analysis, all the possible scenarios which could take place at run-time, to be stored in a look-up table from which to retrieve the correct analysis upon a scenario's occurrence. But this cumbersome activity is in general impossible to conduct at a sufficiently satisfactory level, especially for critical applications subject to strong dependability requirements. Resorting to processing the measurements collected in real operation at a later stage, e.g.

in periodic reviews, may be inadequate as well, since by the time the observations are processed the operational environment may have changed to something not yet seen before.

- Pre-deployment assessment, however, plays an important role in providing a priori knowledge about how the system is expected to operate, especially if the simulated environment is "close" to the operational environment post-deployment, and to take appropriate design decisions. Stochastic model-based assessment has been widely recognized as a helpful means to cover this role [6]. Nevertheless, the unavoidable higher chance of inaccurate/unknown model parameters needs to be considered as a weakness that could result in too inaccurate analysis results, thus negatively impacting design decisions.

To contribute to overcome such deficiencies of current methods in assessing dynamic systems, we developed an approach which tries to combine the benefits of both pre-deployment and processing of data obtained from real executions, as illustrated in the following.

3. Pre-deployment stochastic model based analysis: the way to start supporting design decisions

As already mentioned, pre-deployment assessment is a crucial activity to drive the system design towards a realization compliant with the required level for quality of service indicators. In fact, it allows for early detection of design deficiencies, so as to promptly take the appropriate recovery actions, thus significantly saving in money and time with respect to discovering such problems at later stages. Also, it is central to the decision making process among alternative design solutions, again gaining in efficiency and better guarantee to end up with the "right" system. Stochastic model-based approaches are very suited and widely adopted for early prediction of dependability and performance metrics. Research has developed a variety of models, each one focusing on particular levels of abstraction and/or system characteristics, including State-Based Stochastic methods [21]. These last use state-space mathematical models, expressed with probabilistic assumptions about time durations and transition behaviours; a short survey on State-Based Stochastic methods and automated supporting tools for the assisted construction and solution of dependability models can be found in [6].

The pre-deployment assessment part of our proposed method, tailored to dynamic and evolving systems assessment, exploits State-Based Stochastic mod-

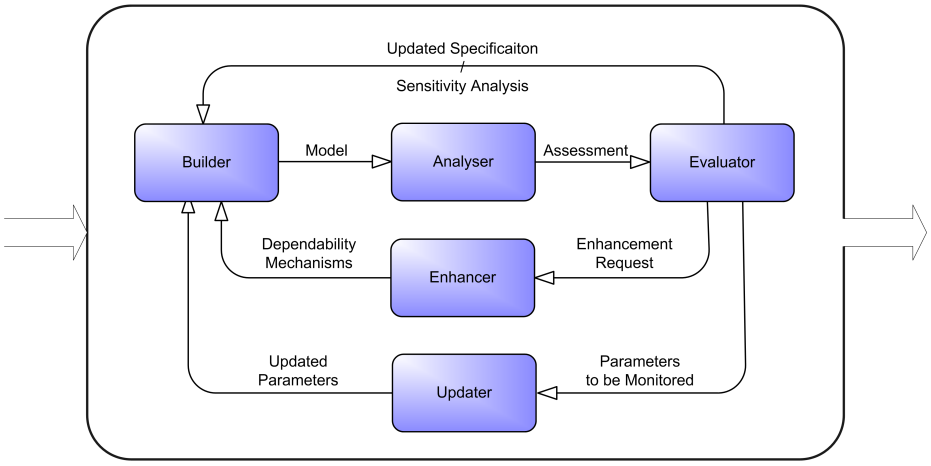


Figure 2: Architecture of the Dependability&Performance (DEPER) Analysis in CONNECT

elling and analysis, which are embedded in an automated process. In the following, we overview the architecture and main functionalities of the Dependability and Performance enabler, introduced in Figure 1 and shortly referred to as DEPER. DEPER supports the automated dependability and performance analysis and has been already partially described in [18, 5]. Figure 2 illustrates the five main modules composing DEPER, whose activities start from pre-deployment assessment of the generated bridging CONNECTors to subsequent refinements based on run-time observations of real networked systems and CONNECTors executions. A brief summary is provided for each module except for the *Udapter* module, which is fully described in Section 5, when focusing on the integration with monitoring. As already introduced in section 2.1, the DEPER enabler interacts with other enablers in the CONNECT framework to: i) be triggered on the analysis to perform and take in input both the specification of the system to analyse and the metric to assess together with the value required for it. The enablers contributing to this step are Discovery, Learning, Deployment and Synthesis; ii) provide the feedback from the analysis to Synthesis, which can then proceed with the deployment of the CONNECTor or refine it according to the feedback.

Builder The Builder module takes in input the specification of the CONNECTed system. This specification is given as Labelled Transition Systems (LTSs) annotated with non-functional information necessary to build the dependability and performance model of the CONNECTed system. Annotations include, for each

labelled transition, the following fields: *time to complete*, *firing probability*, and *failure probability*.
The module produces in output a dependability and performance model of the CONNECTed system suitable to assess the given dependability and performance requirements. Such model is specified with a formalism suitable to describe complex systems that have probabilistic behaviour, e.g., stochastic processes.

Analyser The Analyser module takes in input the dependability and performance model from the Builder module and the dependability and performance properties required by the NSs from Discovery/Learning. These requirements are expressed as *metrics* and *guarantees*. Metrics are arithmetic expressions that describe how to obtain a quantitative assessment of the properties of interest of the CONNECTed system. To allow for automated assessment, they are expressed in terms of transitions and states of the LTS specification of the NSs. Guarantees are boolean expressions that are required to be satisfied on the metrics. The module extends the received model with reward functions suitable to quantitative assessment of the metrics of interest. Then, it makes use of a solver engine to produce a quantitative assessment of the dependability and performance metrics.

Evaluator The Evaluator module is in charge of checking whether the analysis results match with the guarantee, as requested by the networking systems willing to communicate, or not. Evaluator informs Synthesis about the outcome of the check and, in case of mismatch it may receive back a request to evaluate if enhancements can be applied to improve the dependability or performance level of the CONNECTed system, namely:

a) To take into account an alternative CONNECTor deployment (e.g., a deployment that uses a communication channel with lower failure rate). A new analysis is triggered, considering the updated specification of the CONNECTor.

b) Enhance the specification of the CONNECTor by including dependability mechanisms, which are counter-measures to contrast failure modes affecting performance and/or dependability metrics (e.g., a message retransmission technique). Such mechanisms are then applied by the Enhancer module to model elements that are considered weak from the point of view of the metric under assessment.

Instead, in case the analysis results provided by Analyser match with the guar-

antee, the CONNECTor's design is considered satisfactory and ready to be deployed, thus terminating the pre-deployment analysis phase. However, because of possible inaccuracy of model parameters due to potential sources of uncertainty dictated by the dynamic and evolving context, Evaluator also instructs the Updater module about the events to be observed on-line by the Monitor enabler. Collection of such events allows to determine whether a new analysis needs to be performed, to properly adapt to changes (or unforeseen circumstances), as better detailed later in Section 5, when focusing on integration between model-based analysis and on-line monitoring.

Enhancer The Enhancer module is activated by Evaluator when the guarantees are not satisfied and Synthesis makes a request to enhance the CONNECTor with dependability mechanisms. Enhancer is instructed by the Evaluator module with indications about how to select the dependability mechanism to try and to which elements of the original model the mechanism has to be applied. Then, it performs the following actions: (i) selects the dependability mechanisms that can be employed, among those available in the category indicated by Evaluator; (ii) instructs the Builder module on the application of the selected dependability mechanism in the CONNECTed system model, in accordance with indications from Evaluator, and triggers a new analysis. At the end of this new analysis, Evaluator verifies whether the enhanced CONNECTor fulfills the dependability and performance requirements. If yes, Evaluator informs the Synthesis enabler about the mechanism to add to the CONNECTor design and the DEPER's support to the design of this CONNECTor is completed. Otherwise, Enhancer makes a further attempt with the next dependability mechanism (if available), according to some internal predefined policies about the rank of available mechanisms and about how to apply them to model elements provided by Evaluator, and a new cycle with Builder, Analyser and Evaluator is repeated. This loop ends either when a successful mechanism is found, or when all the mechanisms are exhausted. A library of models for triggering the generation of typical dependability mechanisms suitable to contrast two typical classes of failure modes that may happen during interactions has been defined and implemented (see [19]). Given the focus of this paper on the adaptation of model-based analysis through on-line observations, these mechanisms and related models will not be further treated.

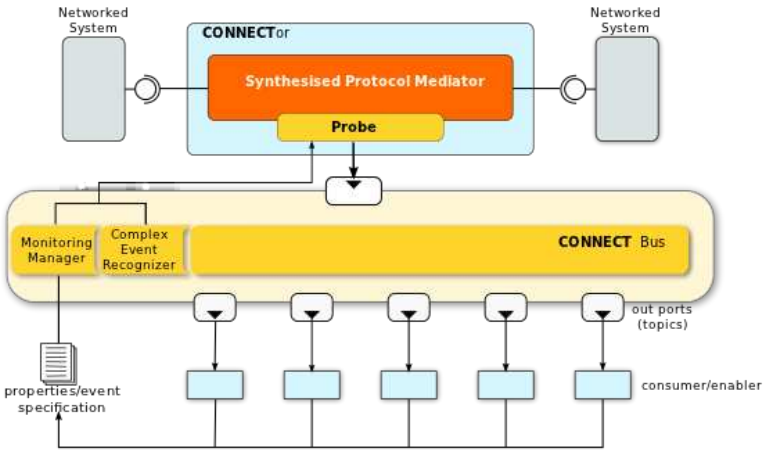


Figure 3: GLIMPSE architecture

4. The on-line view: incremental accumulation of observations through monitoring

The best way to provide a valid bridge between the observed system and the pre-deployment analyser is to insert a new layer, the monitoring layer, able to gather and filter information useful to the dependability and performance analyses. Indeed, monitoring has been used for on-line dependability analysis since the advent of debuggers in the sixties.

In CONNECT we have developed a modular, flexible and lightweight monitoring infrastructure, called GLIMPSE[2]. Although expressly conceived for use in CONNECT, GLIMPSE infrastructure, shown in Figure 3, is totally generic and can be easily applied to different contexts. To provide a better communication decoupling, we adopted a publish-subscribe communication paradigm.

The lowest level of the monitoring is represented by the probe deployed into the CONNECTor; this probe monitors the messages exchanged among the NSs involved into the communication, possibly applying a local filter in order to decrease the amount of messages sent on the CONNECT bus. Note that such probes are non intrusive data collectors (proxies), i.e., they have no effect on the order and timing of events in the application and do not generate overhead on the communication

[2]Glimpse is an acronym for Generic fLexIble Monitoring based on a Publish-Subscribe infrastructurE.

or on the interacting services.

The second layer of the monitoring infrastructure is represented by the information consumers, the entities interested to obtain the evaluation of a non-functional property or interested to receive notification of occurrences of events/exceptions that may occurs into the CONNECTor.

With specific reference to this paper purposes, the Manager module is in charge to manage all the communication between the DEPER enabler (yet another consumer in the Monitor vision) and the Complex Event Processor (CEP). It analyses the request message sent from DEPER and instruments the CEP. The message sent from DEPER enabler contains one or more rules related to nf-properties requirements that the monitor enabler must verify. This message is structured following a generic XSD schema (See listing: 1); we chose such standard format in order to easily allow the replacement of the CEP with any other one that from time to time can be considered more specific or efficient for usage.

The XML generated with this schema contains all the necessary information to interact with the specific knowledge-base used. In particular, into the field RuleName of the XML, DEPER will put the name of the request. The content of the RuleBody field is a rule, written using the *Drools* rule syntax that will be loaded on the GLIMPSE knowledge base. Drools is a well-known rule engine based on Charles Forgy's Rete algorithm [14].

On an event-based monitoring infrastructure, as GLIMPSE, the gathered information is provided in form of events. An event is an atomic description, a smaller part of a larger and more complex process at application level. In CONNECT, an event represents a method invocation on a remote web service: the invocation, coming from the producer to the consumer, is captured when it comes through the CONNECTor, encapsulated into a ConnectBaseEvent object, and sent through the CONNECT bus.

The detailed structure of a ConnectBaseEvent is described in Figure 4.

For completeness we note that, to provide a more abstract generation of a rule for monitoring non-functional properties, we have developed a Property Meta-Model (PMM) [17], from which users can generate their own rule model, so that, using a model-driven approach, this can then be translated directly to the desired/-more performant/available CEP language. We leave the metamodel outside the scope of the present paper.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <schema xmlns="http://www.w3.org/2001/XMLSchema"
3    targetNamespace="http :// labse . isti . cnr . it /glimpse/xml/ComplexEventRule"
4    xmlns:tns="http :// labse . isti . cnr . it /glimpse/xml/ComplexEventRule"
5    elementFormDefault="qualified">
6      <element name="ComplexEventRuleActionList"
7       type="tns:ComplexEventRuleActionType" />
8
9      <complexType name="ComplexEventRuleActionType">
10       <sequence>
11         <element name="Insert" type="tns:ComplexEventRuleType"
12          maxOccurs="unbounded" minOccurs="0" />
13         <element name="Delete" type="tns:ComplexEventRuleType"
14          maxOccurs="unbounded" minOccurs="0" />
15         <element name="Start" type="tns:ComplexEventRuleType"
16          maxOccurs="unbounded" minOccurs="0" />
17         <element name="Stop" type="tns:ComplexEventRuleType"
18          maxOccurs="unbounded" minOccurs="0" />
19         <element name="Restart" type="tns:ComplexEventRuleType"
20          maxOccurs="unbounded" minOccurs="0" />
21       </sequence>
22     </complexType>
23     <complexType name="ComplexEventRuleType">
24       <sequence>
25         <element name="RuleName" type="string" maxOccurs="1" minOccurs="1" />
26         <element name="RuleBody" type="string" maxOccurs="1" minOccurs="0" />
27       </sequence>
28       < attribute  name="RuleType" type="string" />
29     </complexType>
30 </schema>
```

Listing 1: The Complex Event Rule XSD

Figure 4: The ConnectBaseEvent Interface

5. Off-line and on-line integrated: the way to adapt assessment under uncertainty/evolution

After having introduced the pre-deployment and run-time analysis methods under development, we focus here on their synergic usage, through which adaptive assessment is pursued. Basically, the dynamicity and evolution of the targeted environment lead to potential sources of uncertainty, which undermine the accuracy of the off-line analysis. To cope with this issue, adaptive dependability assessment is investigated, which exploits run-time monitoring to re-calibrate and enhance the dependability and performance prediction along time. In brief, the picture of the synergic usage is the following. At design time, stochastic model-based analysis is performed as a pre-deployment method to support the synthesis of a CONNECTor suitable to allow interoperability among the systems willing to connect under required dependability and performance levels. While the prediction so obtained plays an important role in guiding the building of the CONNECTor, it might suffer from unacceptable inaccuracy because of possibly limited knowledge at analysis time or successive context evolution. Through monitoring properly selected events at run-time and collecting them along several executions, we can identify changes that require to be accounted for by a new iteration of the model-based analysis.

5.1. Updater

As shown in Figure 2, Updater is the module of the DEPER architecture in charge of interacting with the Monitor enabler to refine the accuracy of model parameters through on-line observations. Inaccuracy of the non-functional values used in the off-line analysis at CONNECTor design time is mainly due to two possible causes: i) limited knowledge of the NSs characteristics acquired by DEPER/Discovery enablers; ii) evolution along time of the NSs, as naturally accounted for in the CONNECT context.

Updater receives inputs from both internally to DEPER (from the Evaluator module) and externally (from the Monitor enabler).

For each CONNECTor ready to be deployed, the Updater module receives from the Evaluator module the model parameters to convey to the Monitor enabler for run-time observations. The parameters received from the Evaluator are obtained through a sensitivity analysis that aims to understand which elements of the CONNECTed system have highest impact on the dependability and performance measure.

From the Monitor enabler, the Updater module receives a continuous flow of data of the parameters under monitoring relative to the different executions of the CONNECTor. Accumulated data are processed through statistical inference techniques. If, for a given parameter, the statistical inference indicates a discrepancy between the on-line observed behaviour and the off-line estimated value used in the model, a new analysis is triggered by instructing the Builder module to update the CONNECTed system model. To improve on efficiency, the Updater module could receive indications not only on the parameters to be monitored, but also on a range of values for each of them, thus setting the variation interval within which the already performed analysis is subject to negligible modifications. Then, should the new values determined via inference techniques on on-line collected data be outside the reference range values, the consequence is that the synthesised CONNECTor does not meet anymore the stated requirements and re-adjustments at synthesis level are necessary. Of course, the efficiency gained in avoiding repetitions of the analysis triggered by Updater has to be compared with the additional effort necessary at pre-deployment time to assess the ranges for the parameters values via sensitivity analysis. In the current prototype implementation of DEPER, range values have been not accounted for and left as a future extension of the enabler.

The activity diagram that describes the Updater phase is shown in Figure 5.

As reported in [25], methods of statistical inference applied to a collection of elements under investigation (called *population*), allow to estimate the characteristics of the entire population. In our case, the collection of values relative to each

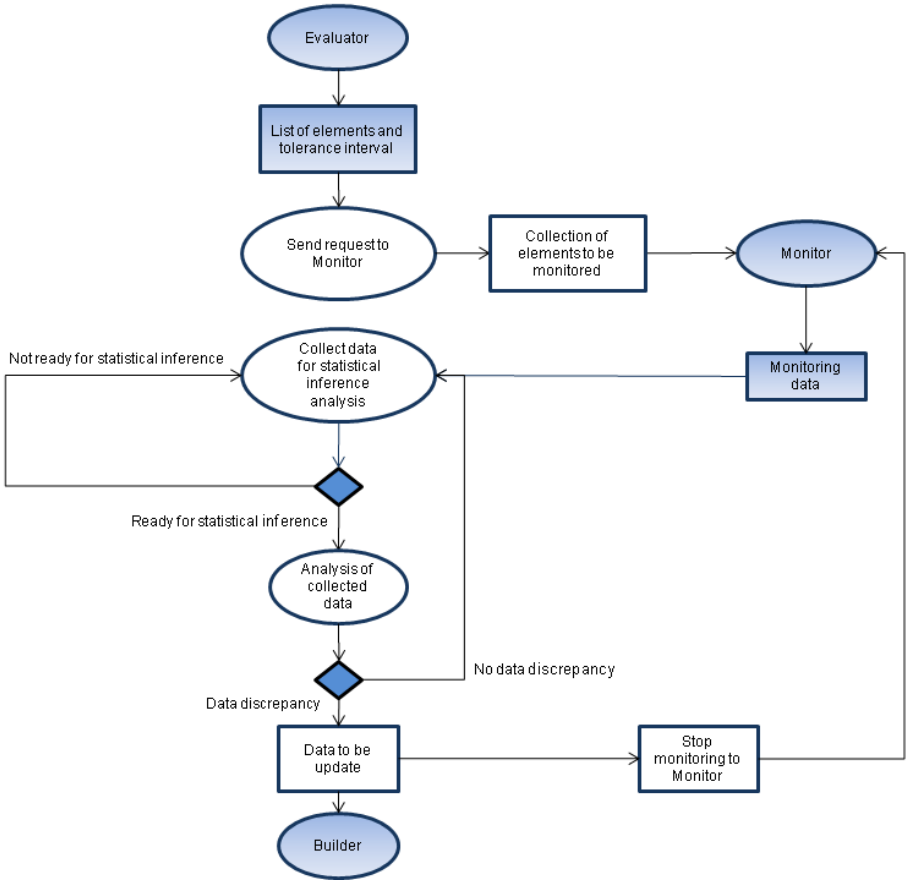


Figure 5: The Updater activity diagram

parameter under monitoring constitute a subset of the population (called *sample*) to which such techniques are applied.
Parameter estimation is the process by which it is possible to get information, from the observed sample, in order to assign a value (*point estimate*) to the parameter or a set of values (*interval estimate*). The sampling process represents a significant problem, because it is unknown which is the representative sample size (n). It seems intuitive that the precision of the estimates increases with n. On the other hand increasing n could lead to excessive increase of time and costs.

The methods of parameter estimation rarely produce a point estimate of the desired parameter which coincides with the actual value. Therefore, it is often

preferred to find an interval estimate, called *confidence interval* Δ, which contains the real value of the parameter under analysis with a *confidence level* α. The size n of the random sample affects the confidence interval, therefore it is possible to determine the value of n based on the confidence interval:

$$n \geq \left\lceil \left(\frac{z_{\alpha/2} S^2(n)}{\Delta} \right)^2 \right\rceil \tag{1}$$

where the value of $z_{\alpha/2}$ is tabulated. When the sample size is relatively small ($n < 30$), we can use the Student *t* distribution [25].
To evaluate the sample size n we encounter two difficulties:

1. S^2 is not known in advance;
2. $t_{n-1;\alpha/2}$, which can be read from a table, depends on n.

These difficulties can be solved by the following two points:

1. using an assumed value of the variance, indicated by S^{*2}, from pilot investigations;
2. using an iterative algorithm, to evaluate n using from time to time the degrees of freedom obtained at the previous step. The stop condition of the algorithm is reached when the result of two successive steps is the same.

The iterative algorithm proceeds as follows:

1. $n_0 = \infty$ (initialisation);
2. $n_1 = \left(\frac{2 \cdot t_{n_0;\alpha/2}}{\Delta} \right)^2 \cdot S^{*2}$;
3. $n_2 = \left(\frac{2 \cdot t_{n_1;\alpha/2}}{\Delta} \right)^2 \cdot S^{*2}$;
4. ...
5. until the last two results are the same.

Following this approach and considering fixed values of confidence interval and confidence level, we are able to define the sample size that the Monitor enabler has to send to the DEPER enabler in order to evaluate the monitored data.

5.2. Integration and interaction

The interaction between DEPER and Monitor can be analysed through a simple sequence diagram shown in Figure 6, where we intentionally left out system start-up operations.

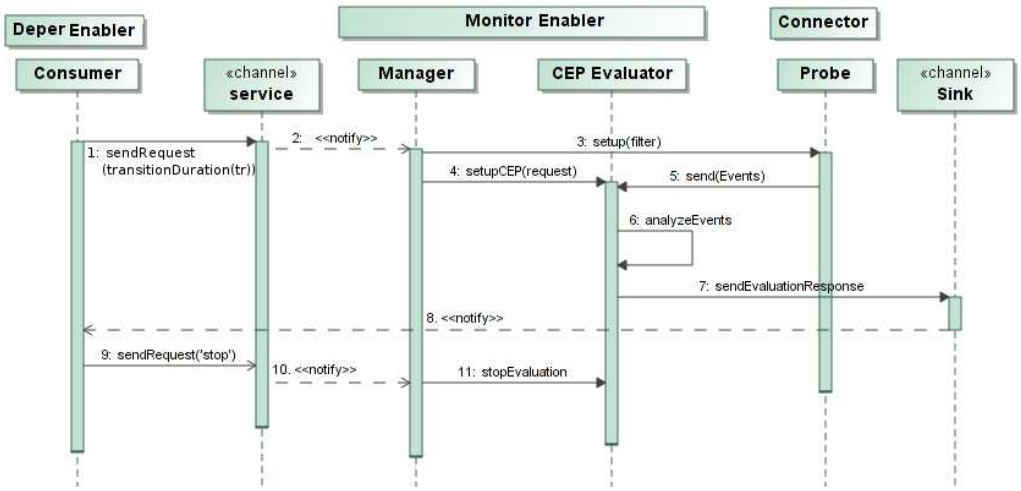


Figure 6: The interaction between DEPER and Monitor

In detail, DEPER and Monitor interact by using a Publish/Subscribe protocol. The interaction starts when the DEPER enabler sends a JMS message whose payload contains an XML object rule generated using ComplexEventRule classes (as explained in Section 4).

Whenever Monitor receives a request message on the service channel, a new channel dedicated to the requesting enabler (DEPER in this case) is set up to communicate occurrences of the requested pattern.

Once the CONNECTor is deployed, data (events) derived from real executions are sent by the probe to the CONNECT bus. The Monitor enabler gathers those events and using the CEP component, correctly instructed through the ComplexEventRuleAction sent by the DEPER enabler, tries to infer one or more of the patterns on which the DEPER enabler is subscribed.

Upon occurrence of a relevant event the notification to the DEPER enabler is enacted by sending a JMS Message on the dedicated channel created on purpose in the initial phase of the communication (see Figure 6) on which payload is a ComplexEventResponse object (see Listing 2).

The DEPER enabler, in turn, performs a statistical analysis of the monitored observations and uses such information to check the accuracy of the model analysed before deployment. If the model parameters are found to be inaccurate, DEPER updates the model with the new values, and performs a new analysis. If the new analysis evidences that the deployed CONNECTor needs adjustments, a

```
1  <?xml version="1.0" encoding="UTF−8"?>
2  <schema xmlns="http://www.w3.org/2001/XMLSchema"
3    xmlns:tns="http :// www.example.org/ComplexEventResponse/"
4    targetNamespace="http :// www.example.org/ComplexEventResponse/"
5     attributeFormDefault =" qualified ">
6    <element name="ComplexEventResponseList" type="tns:ComplexEventResponse" />
7     <complexType name="ComplexEventResponse">
8         <sequence>
9           <element name="RuleName" type="string" maxOccurs="1" minOccurs="1" />
10          <element name="NetworkedSystemSource" type="String" maxOccurs="1" minOccurs="1" />
11          <element name="ResponseKey" type="string" maxOccurs="1" minOccurs="1" />
12          <element name="ResponseValue" type="string" maxOccurs="1" minOccurs="1" />
13        </sequence>
14    </complexType>
15 </schema>
```

Listing 2: The Complex Event Response XSD

new synthesis–analysis cycle starts.

6. Case-study

In this section, we show how the integration between DEPER and GLIMPSE can be exploited in the context of a demonstrative scenario.

6.1. Terrorist alert scenario

We consider the CONNECT Terrorist Alert scenario [9], depicting the critical situation that during a show in the stadium, the control center spots one suspect terrorist moving around. The alarm is immediately sent to the Police.

Policemen are equipped with ad hoc handheld devices which are connected to the Police control center to receive command and documents. Precisely, the policemen can share documents with the Police control center and with other policemen through a *SecuredFileSharing* application, for example a picture of a suspect terrorist.

Unfortunately, the suspect is put on alert from the police movements and tries to escape, evading the Stadium.

Within such an emergency situation, we focus on the case that a policeman that sees the suspect running away can dynamically seek assistance to capture him from civilians serving as private security guards in the zone of interest. To get help in following the moves of the escaping terrorist and capturing him, the policeman sends to the civilian guards an alert message in which one picture of the suspect is distributed.

The guards are equipped with smart radio transmitters which run an *EmergencyCall* application. This transmission follows a two steps protocol. We assume in fact that the guards that control a zone are CONNECTed in groups, and that for each group there is a Commander on duty. The protocol followed in the *EmergencyCall* application is that a request message is first sent from the guards control center to the Commander. As soon as the Commander replies with an acknowledgement of receipt, a message with details of the emergency is forwarded to all security guards. On correct receipt of the alert, each guard's device automatically sends an ack to the control center.

SecuredFileSharing

- The peer that initiates the communication (hereafter denominated the *coordinator*) sends a broadcast message (`selectArea`) to selected peers (the Police control center or policemen) operating in a specified area of interest. In the SecuredFileSharing application, the coordinator can be either the Police control center or a policeman.
- The selected peers reply with an `areaSelected` message.
- The coordinator sends an `uploadData` message to transmit confidential data to the selected peers.
- Each selected peer automatically notifies the coordinator with an `uploadSuccess` message when the data have been successfully received.

EmergencyCall

- The guards control center sends an `eReq` message to the commanders of the patrolling groups operating in a given area of interest.
- The commanders reply with an `eResp` message.
- The guards control center sends an `emergencyAlert` message to all guards of the patrolling groups; the message reports the alert details.
- Each guard's device automatically notifies the guards control center with an `eACK` message when the data has been successfully received and a timeout

is triggered after a time interval if not all guards sends back the `eAck` message. The timeout represents the maximum time that the CONNECTor can wait for the `eAck` message from the guards.

The two applications, *SecuredFileSharing* and *EmergencyCall* in this scenario represent the two Networked Systems, which are not a priori compatible. Hence, to allow a Policeman and the guards in the zone where the suspect has escaped to communicate we need to synthesise on-the-fly a CONNECTor. The needed mappings are shown in Figure 7 and briefly summarised below.

CONNECTor

- The `selectArea` message of the policeman is translated into an `eReq` message directed to the commander of the patrolling group operating in the area of interest.
- The `eResp` message of the commander is translated into an `areaSelected` message for the policeman.
- The `uploadData` message of the policeman is translated into a multicast `emergencyAlert` message.
- The `eACK` messages automatically sent by the guards' devices that correctly receive the `emergencyAlert` message are collected and then translated into a single `uploadSuccess` message for the policeman.

6.2. On-line analysis

Taking as a reference the above described scenario, we focus in the following on the basic interactions between DEPER (in particular, its Updater module) and GLIMPSE enablers, performed to exchange requests for monitoring and to gather monitored data. Figure 8 depicts the dependability and performance model of the synthesised CONNECTor built by DEPER at design time, using the SAN formalism [24]. We recall that this model is obtained through automatic transformation from the LTS specification of the networked system, that is the *SecuredFileSharing* and *EmergencyCall* in the considered scenario. The measures

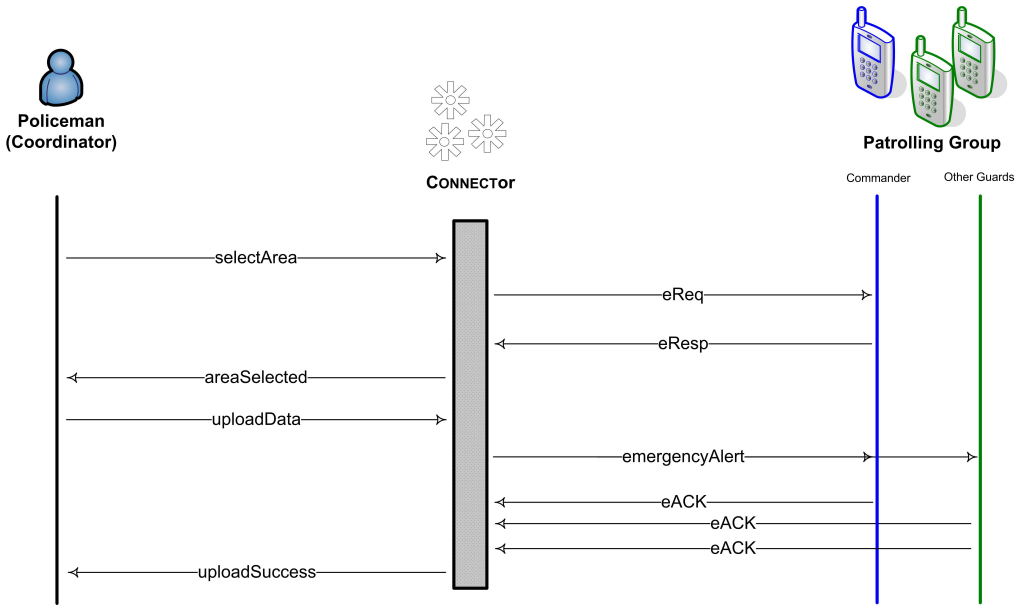


Figure 7: Terrorist Alert Scenario: Sequence diagram of the messages exchange

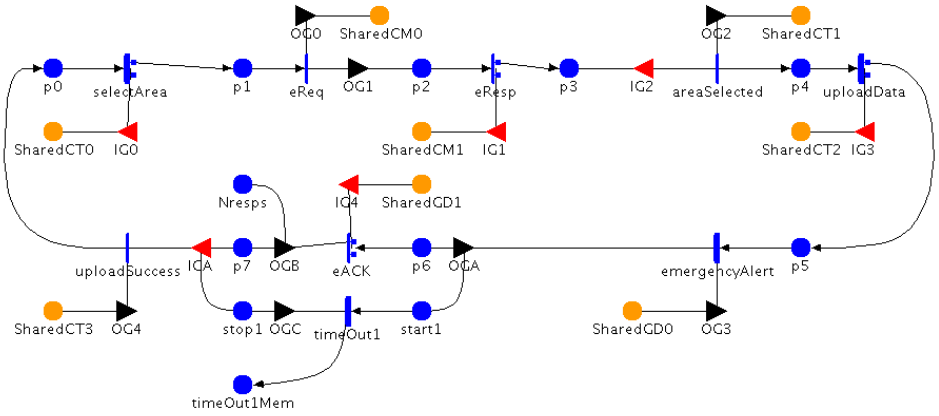


Figure 8: Terrorist Alert Scenario: SAN Model of the CONNECTor

assessed in the evaluation are *latency* and *coverage*. Latency represents a performance indicator and is measured from when the control centre sends the initial request `selectArea` to when it receives `uploadSuccess`. Coverage represents a dependability indicator and is given by the percentage of responses the

control centre receives back within a certain time T. The sensitivity analysis on the impact of model parameters on the assessment of these selected measures revealed that critical parameters to keep under observation on-line via the Monitor enabler are: i) transitions `eReq` and `eResp`, for the latency measure, and ii) the transition `emergencyAlert` for the coverage measure. These model parameters represent the duration of the transitions executed by the NS requesting the communication. Refining the pre-analysis knowledge on the values assumed for such parameters by real observations constitutes a fundamental step in enhancing the accuracy of the analysis results. In fact, should the initial forecast for these parameters deviate from what is evidenced through repeated executions, a new analysis round needs to be triggered to understand whether the dependability and performance requirements are still met by the CONNECTed system.

An example of request message sent by DEPER to GLIMPSE, in order to trigger the monitoring of the critical transition for latency aspects, is shown in the Listing 3.

The GLIMPSE infrastructure, more specifically its Manager component, receives the DEPER requests and sets up the ComplexEventProcessor with the provided rule.

The events flowing in from Probes are structured in a `ConnectBaseEvent` object (see Figure 4), that provides all the necessary informations for an accurate pattern recognition.

According to the scenario, the CONNECTor sends an `eReq` message to the commanders of the patrolling groups operating in a given area of interest.

The event generated from the Probe instrumented into the peer software component is shown in Figure 9 and flows in into the GLIMPSE infrastructure stream of events.

When the commanders reply, another event is fired and sent on the CONNECT bus, the `eResp` event.

The rule `computation time` (lines (20-28) in Listing 3) uses the timestamp impressed into the two different events to infer latency, and matches the parameters: `connectorID`, `sequenceID`, `ConnectorInstanceID`, and `ConnectorInstanceExecutionID` to check that the events are generated from the same CONNECTor. This rule allows to calculate the latency (line 35) and to provide it to DEPER (line 40-41).

Indeed, the rule `pending request` in the Listing 3, (lines (48-54)), computes the number of incoming requests into the CONNECTor and provides it to DEPER.

We first consider the steps to refine the accuracy of the *failure probability*

```
<?xml version="1.0" encoding="UTF-8"?>
<ComplexEventRuleActionList>
  <Insert RuleType="drools">
  <RuleName>Computation Time</RuleName>
  <RuleBody>
  declare ConnectBaseEventImpl
      @role(event)
      @timestamp(timestamp)
  end
  declare SatisfiedRequest
      duration : float
      incoming : SimpleEvent
      outcoming: SimpleEvent
  end
  rule "computation time"
  no-loop
  salience 999
  dialect "java"
  when
    $aEvent:ConnectBaseEventImpl(this.data=="eReq",
          this.getConsumed == false);
    $bEvent:ConnectBaseEventImpl(this.data=="eResp",
      this.getConsumed == false,
      this.getConnectorID == $aEvent.getConnectorID,
      this.getConnectorInstanceID == $aEvent.getConnectorInstanceID,
      this.getConnectorInstanceExecutionID ==
      $aEvent.getConnectorInstanceExecutionID,
      this after $aEvent);
  then
    $aEvent.setConsumed(true);
    $bEvent.setConsumed(true);
    SatisfiedRequest sr = new SatisfiedRequest();
    sr.setIncoming($aEvent);
    sr.setOutcoming($bEvent);
    sr.setDuration(DroolsUtils.latency($aEvent.getTimestamp(),
                $bEvent.getTimestamp()));
    insert(sr);
    retract($aEvent);
    retract($bEvent);
    ResponseDispatcher.NotifyMe(drools.getRule().getName(),
            "DePer module", sr.getDuration());
  end
  rule "pending request"
  no-loop
  salience 999
  dialect "java"
  when
    $total : Number()
    from accumulate($nEvent : ConnectBaseEventImpl(data=="eReq")
          from entry-point "DEFAULT",
          count($nEvent))
  then
    ResponseDispatcher.NotifyMe(drools.getRule().getName(),
          "DePer Module", "PENDING: " + $total);
  end
    </RuleBody>
 </Insert>
</ComplexEventRuleActionList>
```

Listing 3: Sample Request from DEPER enabler to Monitor

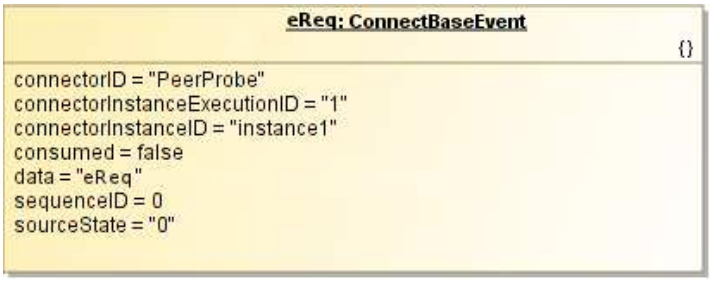


Figure 9: The eReq event sent from the PeerProbe

of the communication channel between the *EmergencyCall* application and the CONNECTor. In order to get statistically significant estimations from the analysis of the data gathered from the Monitor, we fixed the confidence level to 95%, the confidence interval to 0.1, and the variance to 0.01. We accumulated data generated from several executions of the CONNECTor in scenario's configurations where the number of guards was varying. In each configuration, executions have been performed until the mean value of `emergencyAlert` message occurrences notified by Monitor stabilizes within the assumed confidence interval. From such mean value, the mean failure probability we are interested in is obtained as $1 - (number\ of\ guards / number\ of\ emergencyAlert)$. Then, applying the iterative algorithm presented in Section 5.1 to the mean failure probability for each scenario's configuration, the overall mean failure probability is obtained. Table 1 summarizes the data involved in this experiment to obtain (as the average of the values reported in the last column) the refined value of 0.1416 for the parameter under observation. The value assumed during pre-deployment dependability analysis was 0.05, a clearly divergent value calling for a new evaluation of the coverage measure.

Number of Guards	Occurences of ``emergencyAlert''	Failure probability
22	18.94	0.139
33	28.84	0.126
44	37.96	0.137
55	46.54	0.154
110	93.27	0.152

Table 1: Elaboration of data from Monitor to update the failure probability parameter

Figure 10 shows the trend of the *coverage* (on the *y* axis) for different values

of the failure probability (on the x axis). Also, the threshold coverage line as specified in the requirement (set to the value 0.8) is reported. Not surprisingly, as the failure probability increases, coverage decreases. The coverage value obtained through the pre-deployment analysis is 0.9, fully satisfying the requirements. But the coverage value after updating the failure probability parameter is 0.73, not a satisfactory value anymore. The CONNECTor needs to be enhanced; therefore DEPER informs the Synthesis enabler about the analysis results and appropriate actions are taken by Synthesis (typically, a new CONNECTor is synthesised).

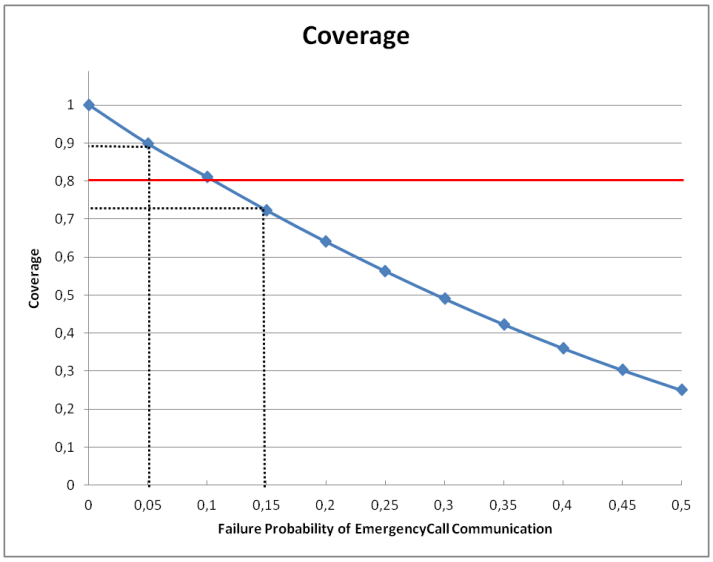


Figure 10: Trend of Coverage as a function of failure probability of the EmergencyCall channel

Now, we move to the steps to refine the accuracy of the model parameters critical for the assessment of the *latency* indicator. They are the execution time of the model transitions `eReq` and `eResp` in Figure 8. These transition execution time are represented by an exponential distribution, with rate 1. Similarly to the previous case of coverage, executions have been performed and the time durations of the transitions under observations collected from Monitor. Table 2 summarizes the mean values for the time duration of the two transitions (in time units). Through the *probability plotting paper* method [4], it is then possible to estimate the actual value of the distribution rate, that results to be 0.89.

Figure 11 shows the trend of *latency* (on the y axis) at increasing values of *Timeout* (on the x axis). The latency threshold specified in the requirements (30 time units) is also depicted. The figure includes three plots, corresponding to:

Transition	Timing duration
eReq	9.650855
eResp	10.647591

Table 2: Timing values from Monitor

(i) the results of the pre-deployment analysis; (ii) the results of the analysis after the parameters influencing latency have been updated; and (iii) the results of the analysis after both the parameters influencing latency and coverage have been updated. It is worth noting that latency exceeds the required threshold only when all model parameters under on-line observation have been updated, for values of Timeout bigger than 21 time units.

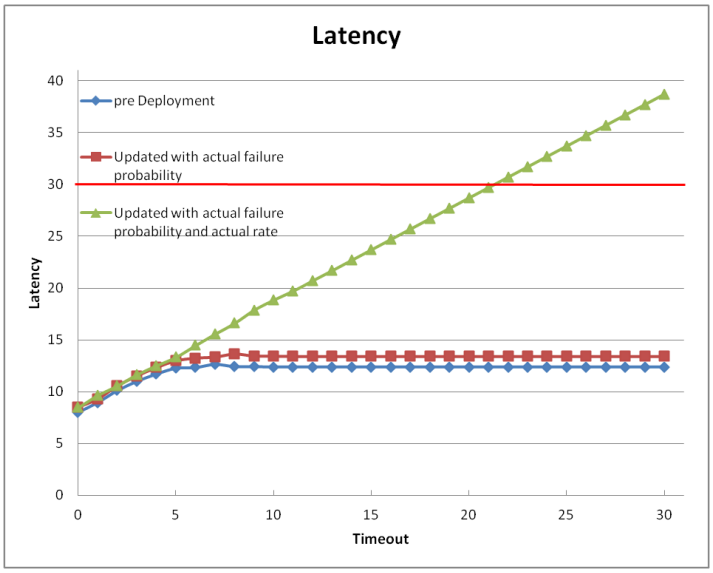


Figure 11: Trend of Latency as a function of Timeout

Similarly, Figure 12 shows the trend of *coverage* (on the *y* axis) at increasing values of *Timeout* (on the *x* axis). As in the previous figure, we show the pre-deployment analysis results, those of the analysis performed after updating the value of the failure probability, and those relative to the analysis where both coverage and latency related parameters have been updated at run-time. It can be noted that pre-deployment estimation of coverage was too optimistic: if the coverage requirement is set higher than 0.73, the synthesised CONNECTor fails to meet it, whichever be the assumed value for the Timeout.

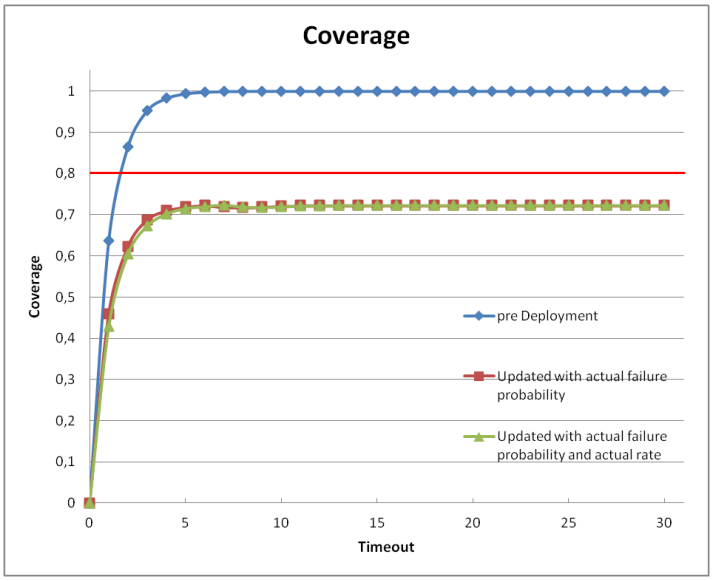


Figure 12: Trend of Coverage as a function of Timeout

7. Related work

This paper addresses the integration between stochastic model-based analysis of dependability and performance and event-based monitoring, in order to meet the needs of adaptive analysis in dynamic and evolving contexts.

Stochastic model-based approaches for quantitative analysis of non-functional properties have been largely developed along the last decades and documented in a huge literary production on this topic. The already cited papers [21, 6] provide a survey of the most popular ones. The choice of the most appropriate type of model to employ depends upon the complexity of the system under analysis, the specific aspects to be studied, the attributes to be evaluated, the accuracy required, and the resources available for the study. The prototype implementation of our DEPER enabler is based on Stochastic Activity Networks (SANs) [24], a variant of the Stochastic Petri Nets class.

With regard to monitoring, various approaches have been recently proposed. Similarly to GLIMPSE, also [22] presents an extended event-based middleware with complex event processing capabilities on distributed systems, adopting a publish/subscribe infrastructure, but it is mainly focused on the definition of a complex-event specification language. The aim of GLIMPSE is to give a more general and flexible monitoring infrastructure for achieving a better interpretabil-

ity with many possible heterogeneous systems.

Another monitoring architecture for distributed systems management is presented in [15]. Differently from GLIMPSE, this architecture employs a hierarchical and layered event filtering approach. The goal of the authors is to improve monitoring scalability and performance for large-scale distributed systems, minimizing the monitoring intrusiveness.

Many works focus on the definition of expressive complex event specification languages. Among them, GEM [16] is a generalized and interpreted event monitoring language. It is rule-based (similar to other event-condition-action approaches) and also provides a tree-bases detection algorithm taking into account communication delay. Also the Snoop language [8] follows an event-condition-action approach supporting temporal and composite events specification but it is especially developed for active databases. A more recent formally defined specification language is TESLA [12]. It has a simple syntax and a semantics based on a first order temporal logic. Some existing open-source event processing engines are Drools Fusion [1] and Esper [2]. They can fully be embedded in existing Java architectures and provide efficient rule processing mechanisms. In our prototype we used Drools because ServiceMix offers it as business rule engine.

The focus of our approach is in the combined usage of pre-deployment model-based analysis and run-time observations via monitoring. Preliminary studies that attempt combining off-line with on-line analysis have already appeared in the literature. A major area on which such approaches have been based is that of autonomic computing. Among such studies, in [20], an approach is proposed for autonomic systems, which combines analytic availability models and monitoring. The analytic model provides the behavioural abstraction of components/subsystems and of their interconnections and dependencies, while statistical inference is applied on the data from real time monitoring of those components and subsystems, to assess parameter values of the system availability model. Through on-line monitoring and estimation of system availability, adaptive on-line control of system availability can then be obtained. In [23], an approach is proposed to carry out run-time reliability estimation, based on a preliminary modelling phase followed by a refinement phase, where real operational data are used to overcome potential errors due to model simplifications. The model is based on Discrete Time Markov Chain, and a prototype version of the monitoring system has been implemented, that is initially trained with the reference model and the preliminary reliability estimation, and then uses operational data to compute the on-line reliability level. Our approach aims at proposing a general and powerful evaluation framework, tailored to a variety of dependability and performance metrics, to

meet a wide spectrum of system requirements and adaptation needs.

Another point of strength of our approach is the ability to automate the analysis process along the system lifetime, from the design phase to the operational one, based on transformation rules. Research on definition and development of transformation-based verification and validation environments are being pursued since several years. Providing automatic/automated transformations methods from system specification languages to modelling languages amenable to perform dependability analysis has been recognized as an important support for improving the quality of systems. In addition, it favours the application of verification and validation techniques at industry level, where these methods are not widely used primarily due to the high level of abstractness of the mathematical modelling and analysis techniques. To provide some examples, the Viatra tool [11] automatically checks consistency, completeness, and dependability requirements of systems designed using the Unified Modeling Language. The Genet tool [7], based on the theory of regions [13], allows the derivation of a general Petri net from a state-based representation of a system. Our work addresses the transformation from the LTS formalism, as system specification language, to SAN, as dependability modelling language. Since there are some steps in common with the Genet tool and related theory, we partially reused available results from this previous study in our prototype implementation.

8. Final discussion and conclusions

In this paper, we have tackled the challenge of dependability and performance analysis in dynamic and evolving systems, whose peculiarities make traditional methods largely inadequate. Our proposal to cope with the issues raised in the addressed context resorts to integrate pre-deployment stochastic model-based analysis with run-time monitoring, to achieve adaptive dependability assessment through re-calibration and enhancement of the dependability and performance prediction along time. The aim of this two-phase analysis framework is twofold. On one side, the stochastic model-based analysis performed at pre-deployment time provides a primary support to the realization of the "rightest" system, given the partial knowledge about the involved subsystems and environment conditions. However, the resulting unavoidable potential inaccuracy on the prediction so obtained could lead to more or less severe consequences if awareness about it is never acquired. This is the point where, on the other side, monitoring provides its contribution, by observing events which help to enhance the previously performed analysis. In fact, through monitoring properly selected events during execution and collect-

ing them along multiple executions, we can identify changes that require to be accounted for by a new iteration of the model-based analysis.

This pre-deployment and run-time integrated framework is proposed as a general, automated approach to fulfill the dependability and performance assessment needs in dynamic and evolving contexts.Therefore, although not novel in its basic principles, the work done so far constitutes an important step towards the definition of an automated and adaptive process to provide dependability and performance analysis accounting for modern and future application needs.

At the time of writing, prototypes of DEPER, GLIMPSE and their integration are already available from the CONNECT project. However, additional effort is needed to fully embed in them the many potentialities that we foresee could be offered by the approach. Especially, techniques would be desirable to balance between time to produce results and their accuracy. For instance, in the automatic generation of the dependability and performance model from the specification of the CONNECTed system, techniques could be sought able to optimise the model on the basis of the specific metrics that needs to be assessment. Also, compositional solution methods for the dependability and performance model would be highly attractive, possibly reusing partly solved model, e.g., when the synthesised CONNECTor is derived as specialisation of an already existing CONNECTor that has already been analysed, or when already analysed dependability mechanisms are introduced in the dependability model.

In the case study, we considered that the monitor can send information about basic events, such as counting the occurrences of an event or the duration of model transitions. In view of better exploiting the functionalities of the GLIMPSE monitoring infrastructure, which is able to observe complex events as aggregation of elementary ones, GLIMPSE could be instrumented to evaluate final properties of interest, like coverage. Then, the comparison with pre-deployment assessment made through DEPER would become a powerful cross-validation operation, reinforcing the confidence in the design-time forecast, or revealing inadequacy of the assumed model parameter (we exclude potential deficiencies in the set-up of the model itself since the dependability and performance models are automatically derived from the LTS specifications).

The approach we illustrated is part of a wider comprehensive framework for providing seamless and "eternal" interoperability in dynamic and evolving contexts by means of on-the-fly emerging middleware. The project leading principle is to be as general as possible and to make the least assumptions on what information and interface the Networked Systems to be connected should expose. In addition to the approach reported here for dependability and performance assess-

ment, the project has developed enablers concerning Discovery, Synthesis, Learning and Deployment of CONNECTors developed according to different technologies as well as support for Security and Trust evaluation. We refer to the project web site for the latest results on the aspects of CONNECT not covered here.

Finally, we would also like to underline that, despite we developed the approach assuming the CONNECT context to make the exposition more concrete, the approach is general and applicable to other contexts sharing the characteristics of evolution and partially known specifications.

Acknowledgements

This work has been partially supported by the European Project CONNECT Grant Agreement No.231167.

References

[1] Drools fusion: Complex event processor. http://www.jboss.org/drools/drools-fusion.html.

[2] Esper: Event stream and complex event processing for java. http://www.espertech.com/products/esper.php.

[3] L. Baresi, C. Ghezzi, and E. Di Nitto. Toward open-world software: issues and challenges. *Computer*, 39(10), 2006.

[4] V. Barnett. Probability plotting methods and order statistics. *Journal of the Royal Statistical Society*, Series C (Applied Statistics) Vol. 24(1):95–108, 1975.

[5] A. Bertolino, A. Calabrò, F. Di Giandomenico, and N. Nostro. Dependability and performance assessment of dynamic connected systems. In M. Bernardo and V. Issarny, editors, *Formal Methods for Eternal Networked Software Systems*, volume 6659 of *LNCS*, pages 350 – 392. Springer, 2011.

[6] A. Bondavalli, S. Chiaradonna, and F. Di Giandomenico. Model-based evaluation as a support to the design of dependable systems. In Hassan B. Diab and Albert Y. Zomaya, editors, *Dependable Computing Systems: Paradigms, Performance Issues, and Applications*, pages 57–86. Wiley, 2005.

[7] J. Carmona, J. Cortadella, and M. Kishinevsky. Genet: A tool for the synthesis and mining of petri nets. In *ACSD '09*, pages 181–185, Washington, DC, USA, 2009. IEEE Computer Society.

[8] S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Data & Knowledge Engineering*, 14(1):1–26, 1994.

[9] CONNECT Consortium. Deliverable 6.1 – Experiment scenarios, prototypes and report Iteration 1, 2011.

[10] ReSIST Consortium. Eu project resist: Resilience for survivability in ist. deliverable d13: From resilience-building to resilience-scaling technologies: Directions. Technical report, 2007. http://www.resist-noe.org/.

[11] Gyorgy Csertan, Gabor Huszerl, Istvan Majzik, Zsigmond Pap, Andras Pataricza, Daniel Varro, and Dniel Varr. Viatra - visual automated transformations for formal verification and validation of uml models. In *17th IEEE International Conference on Automated Software Engineering (ASE'02)*, pages 267–270, 2002.

[12] Gianpaolo Cugola and Alessandro Margara. TESLA: a formally defined event specification language. In *Proceedings of DEBS*, pages 50–61, 2010.

[13] A. Ehrenfeucht and G. Rozenberg. Partial (set) 2-structures. Part I: basic notions and the representation problem. *Acta Inf.*, 27(4):315–342, 1990.

[14] Charles Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligences*, 19(1):17–37, 1982.

[15] Ehab Al-Shaer Hussein, Hussein Abdel-wahab, and Kurt Maly. HiFi: A New Monitoring Architecture for Distributed Systems Management. In *Proceedings of ICDCS*, pages 171–178, 1999.

[16] Masoud Mansouri-Samani and Morris Sloman. GEM: a generalized event monitoring language for distributed systems. *Distributed Systems Engineering*, 4(2):96–108, 1997.

[17] Antinisca Di Marco, Claudio Pompilio, Antonia Bertolino, Antonello Calabrò, Francesca Lonetti, and Antonino Sabetta. Yet another meta-model to specify non-functional properties. In Domenico Bianculli, Sam Guinea,

Andreas Metzger, and Andrea Polini, editors, *QASBA*, ACM International Conference Proceeding Series, pages 9–16. ACM, 2011.

[18] P. Masci, M. Martinucci, and F. Di Giandomenico. Towards automated dependability analysis of dynamically connected systems. In *Proc. IEEE International Symposium on Autonomous Decentralized Systems*, pages 139 – 146. IEEE, Kobe, Japan, June 2011.

[19] P. Masci, N. Nostro, and F. Di Giandomenico. On enabling dependability assurance in heterogeneous networks through automated model-based analysis. In *Proc. Third International Workshop, SERENE 2011*, volume 6968 of *LNCS*, pages 78 – 92. Sringer, Geneva, Switzerland, September 2011.

[20] Kesari Mishra and Kishor S. Trivedi. Model based approach for autonomic availability management. In *ISAS 2006*, volume 4328 of *LNCS*, pages 1–16. Lecture Notes in Computer Science, 2006.

[21] David M. Nicol, William H. Sanders, and Kishor S. Trivedi. Model-based evaluation: from dependability to security. *IEEE Transactions on Dependable and Secure Computing*, 1:48–65, January-March 2004.

[22] P.R. Pietzuch, B. Shand, and J. Bacon. Composite event detection as a generic middleware extension. *Network, IEEE*, 18(1):44 – 55, jan. 2004.

[23] K. S. Trivedi R. Pietrantuono, S. Russo. Online monitoring of software system reliability. In *Proc. EDCC '10 - 2010 European Dependable Computing Conference*, pages 209–218. IEEE Computer Society, 2010.

[24] W. H. Sanders and L. M. Malhis. Dependability evaluation using composed SAN-based reward models. *Journal of Parallel and Distributed Computing*, 15(3):238–254, 1992.

[25] K. S. Trivedi. *Probability and Statistics with Reliability, Queueing and Computer Science Applications*. John Wiley & Sons, New York, second edition, 2002.

On-the-Fly Dependable Mediation between Heterogeneous Networked Systems

Antonia Bertolino[1], Antonello Calabrò[1], Felicita Di Giandomenico[1], Nicola Nostro[1], Paola Inverardi[2], and Romina Spalazzese[2]

[1] Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", CNR, Pisa, Italy
[2] Department of Computer Science, University of L'Aquila, Coppito (AQ), Italy
firstname.lastname@isti.cnr.it,
{paola.inverardi,romina.spalazzese}@univaq.it

Abstract. The development of next generation Future Internet systems must be capable to address complexity, heterogeneity, interdependency and, especially, evolution of loosely connected networked systems. The European project CONNECT addresses the challenging and ambitious topic of ensuring eternally functioning distributed and heterogeneous systems through on-the-fly synthesis of the CONNECTors through which they communicate. In this paper we focus on the CONNECT enablers that dynamically derive such connectors ensuring the required non-functional requirements via a framework to analyse and assess dependability and performance properties. We illustrate the adaptive approach under development integrating synthesis of CONNECTors, stochastic model-based analysis performed at design time and run-time monitoring. The proposed framework is illustrated on a case study.

1 Introduction

We live in the Future Internet (FI) era, which is characterized by unprecedented levels of connectivity and evolution. Software systems are increasingly pervasive, dynamic and heterogeneous, and many -even critical- aspects of modern society rely on their continuous availability and seamless interoperability. Ensuring the successful dynamic composition among heterogenous, independently developed Networked Systems (NSs) raises the need of novel computing paradigms, such as the revolutionary approach to *on-the-fly connection* pursued within the European FP7 Future and Emerging Technology Project CONNECT.

CONNECT follows the ambitious goal of enabling *seamless* and *dependable* interoperability among NSs in spite of technology diversity and evolution. The key idea is to compose systems by generating on-the-fly the interoperability solution necessary to assure the connection among the heterogenous NSs both at application and at middleware level. The synthesized solution is called a CONNECTor or also *mediating connector* or *mediator* for short; the system obtained from the composition of the NSs through the CONNECTor is said the CONNECTed System.

Automatically synthesized CONNECTors are concrete emergent entities that mediate the NSs discrepancies, i.e., they translate and coordinate mismatching interaction protocols, actions, and/or data models, allowing applications to interact effectively.

M. José Escalona, J. Cordeiro, and B. Shishkov (Eds.): ICSOFT 2011, CCIS 303, pp. 20–37, 2012.

A synthesized CONNECTor, if it exists, provides by construction a correct solution to functional interoperability among the NSs.

This is however not sufficient: effective interoperability also requires that such on-the-fly CONNECTed systems provide the required non-functional properties and continue to do so even in presence of evolution. The CONNECTors are not a priori guaranteed to provide the desired non-functional properties for the CONNECTed system, thus a suitable and adaptive assessment framework is required. In this paper, we focus on the problem of ensuring the non-functional properties for CONNECTed systems.

Concerning dependability and performance properties, several challenges arise. Off-line, or pre-deployment, assessment can help to take appropriate design decisions by providing a priori feedback about how the system is expected to operate. Nevertheless, the unavoidable high chance of inaccurate/unknown model parameters might result in inadequate analysis results. Moreover, the many possible variations occurring during the system lifetime would require to foresee and analyze all the possible scenarios which could take place at run-time (e.g., to be stored in a look-up table from which to retrieve the correct analysis upon a scenario's occurrence). On the other hand, resorting to processing the measurements collected in real operation at a later stage, e.g. in periodic reviews, may be inadequate, since by the time the observations are processed the operational environment may have changed. Furthermore, in the CONNECT context the above problems are exacerbated because, as said, components are dynamically assembled to satisfy an emergent user goal. In this scenario the only part of the system under control is the synthesized CONNECTor, whereas for the NSs only declarative or learned on-the-fly knowledge can be assumed.

To contribute to overcome the above issues, we have developed an approach which tries to combine the benefits of both pre-deployment and processing of data obtained from real executions. The proposed assessment framework combines stochastic model-based analysis [1] with continuous on-line assessment of non-functional properties through a lightweight flexible monitoring infrastructure, and applies such approach to the on-the-fly CONNECT system into a continuous loop.

In the following we initially provide the context for our approach by introducing the CONNECT architecture (Section 2). Then we introduce the case study that is used to demonstrate the applicability of the integrated analysis framework (Section 3). Mediator synthesis (Section 4), pre-deployment analysis (Section 5) and the run-time monitor (Section 6) are briefly presented, and hence their synergic usage (Section 7), through which adaptive assessment is pursued. Finally we overview related work (Section 8) and draw conclusions (Section 9).

2 The CONNECT Project

Our research is carried out in the context of the FP7 "ICT forever yours" European Project CONNECT[1], belonging to the Future and Emerging Technologies track. As said in the introduction, the ambition of the project is to have eternally functioning systems within a dynamically evolving context.

[1] `http://connect-forever.eu`

In Figure 1 we provide an overview of the CONNECT vision and architecture. In brief, the NSs manifest the intention to connect to other NSs. The *Enablers* are networked entities that incorporate all the intelligence and logic offered by CONNECT for enabling the required connection. We show in schematic form the enablers which are currently part of the CONNECT enabling architecture:

Discovery Enabler: discovers the NSs, catches their requests for communication and initiates the CONNECT process. We tend to make the minimum possible assumptions on the information (called the *affordance*) that NSs must provide;

Learning Enabler: we use active learning algorithms to dynamically determine the interaction behaviour of a NS and produce a model in the form of a labeled transition system (LTS);

Synthesis Enabler: from the models of the two NSs, this enabler synthesizes a mediator component through automated behavioural matching. More details on this enabler are given in Section 4;

Deployment Enabler: deploys and manages the synthesized CONNECTors;

Monitor Enabler: collects raw information about the CONNECTors behaviour, filters and passes them to the enablers who requested them. The CONNECT monitoring infrastructure is further described in Section 6;

DEPER Enabler: this is the enabler assessing dependability and performance properties and is described in detail in Section 5;

Security and Trust Enabler: collaborates with the synthesis enabler to satisfy possible security and trust requirements. It also continuously determines if the requirements are maintained at run-time, by receiving monitoring data from the monitoring enabler. For reasons of space, we do not deal with this enabler in this paper.

All communication among the enablers and with the CONNECTors happens through a message bus, which is currently implemented by a simple message-based communication model (as for instance the Java Messaging Service (JMS)).

In this paper we provide a snapshot of the functioning of CONNECT over the case study introduced in the following section. For space limitation, we focus on the interaction among Synthesis, Dependability&Performance and Monitor, which are highlighted by tick borders in Figure 1. We show first how a dependable CONNECTor is deployed (pre-deployment analysis), and then how, via the feedback obtained through run-time monitoring of the CONNECTor behaviour, CONNECTor adaptation is triggered and managed. In particular, we devise a process for the CONNECTor creation that is supported by powerful infrastructures made available by CONNECT itself. Once the Discovery Enabler discovers new devices, the CONNECT supporting infrastructure starts the computation of a CONNECTor on the fly –if possible. Then, when the intent to communicate is manifested, the CONNECTor -if it exists- is partially computed and is hence concretized. Note that the CONNECTor could not exist because NSs are not compatible and then do not have a way to communicate.

3 Case Study

In this section, we present our running example for presenting how synthesis, analysis and monitoring work in integrated way.

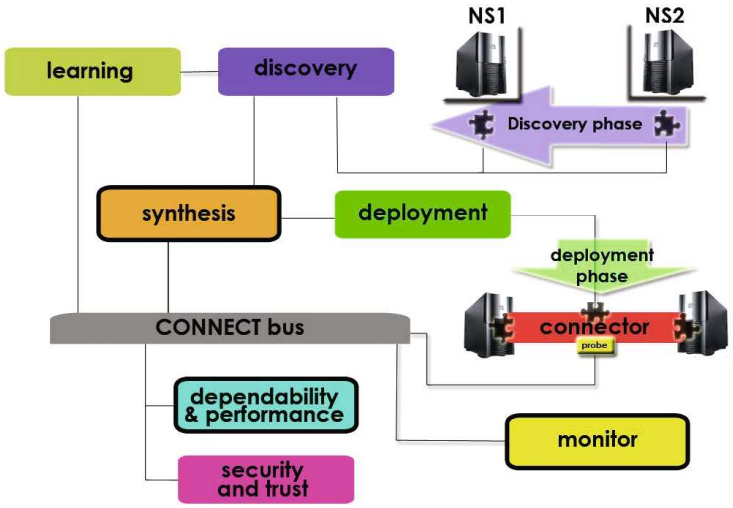


Fig. 1. The CONNECT architecture

3.1 Terrorist Alert Scenario

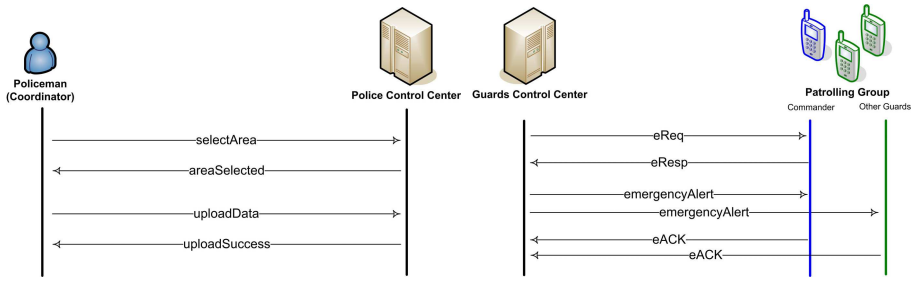
We consider the CONNECT Terrorist Alert scenario [2], depicting the critical situation that during a show in the stadium, the stadium control center spots one suspect terrorist moving around. This emergency situation makes it necessary to exchange information between policemen and security guards patrolling the surroundings of the stadium equipped with heterogeneous applications.

Each policeman can exchange confidential data with other policemen with a *SecuredFileSharing* application. Security guards, on the other hand, exchange information by using another application, denominated *EmergencyCall*. The two applications have the same aim (i.e., enable information exchange), but use different protocols as we describe in the following.

SecuredFileSharing

- The peer that initiates the communication denominated *coordinator* (the Policemen of our example) sends a broadcast message (`selectArea`) to selected peers operating in a specified area of interest (the Police control center of our example).
- The selected peers reply with an `areaSelected` message.
- The coordinator sends an `uploadData` message to transmit confidential data to the selected peers.
- Each selected peer automatically notifies the coordinator with an `uploadSuccess` message when the data have been successfully received or the coordinator can receive an exception.

An example of message flow between a coordinator, i.e., a Policeman, and a Police control center is depicted in Figure 2(a) while the application behaviour of another Policeman is shown in Figure 3. It is worth to notice that, for readability, in the figure



(a) *SecuredFileSharing* Application (b) *EmergencyCall* Application

Fig. 2. Sequence Diagrams of the applications

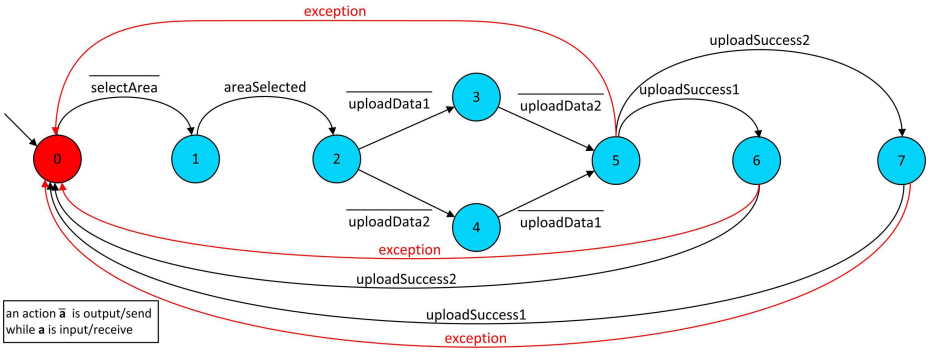


Fig. 3. LTS of the *Policeman* application

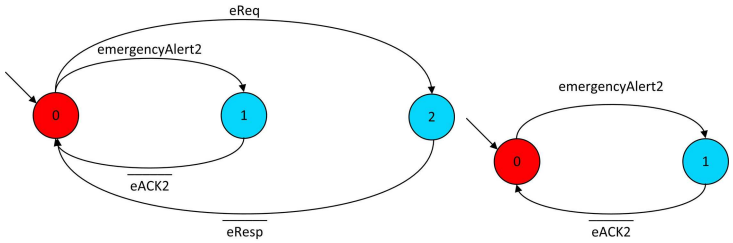
the LTS of the Policeman describes the sending of the broadcast alert to two selected peers while in the experiments we conducted, we used 11 selected peers.

The affordance of the Policeman application includes also the following *non-functional requirement*: she/he should receive the 65% of acknowledgements for the alerts sent within 30 time units, otherwise a failure is reported.

EmergencyCall

- The Guards Control Center sends an `eReq` message to the Commanders of the Patrolling Groups operating in a given area of interest.
- The Commanders reply with an `eResp` message.
- The Guards Control Center sends an `emergencyAlert` message to all Guards of the patrolling groups; the message reports the alert details.
- Each Guard's device automatically notifies the Guards Control Center with an `eACK` message when the data has been successfully received.

The message flow among the Guard control Center, the Commander and the Other Guards is depicted in Figure 2(b). Figure 4(a) shows the LTS of the Commander, and the LTS of the Other Guards is shown in Figure 4(b).



(a) Commander. (b) Other guards.

Fig. 4. LTSs of the *EmergencyCall* application

3.2 CONNECT in the Case Study

With reference to CONNECT architecture (see Figure 1), the two NSs which need to communicate but are not a priori compatible, are the devices implementing the described applications *SecuredFileSharing* and *EmergencyCall*. Hence, to allow a Policeman and the Guards, operating in the zone where the suspect terrorist has escaped, to communicate, CONNECT proposes to automatically synthesize on-the-fly a CONNECTor that can mediate between the two different communication protocols. Such CONNECTor should be able to support the exchange of information, while also fulfilling possible non-functional requirements.

4 Automated Mediator Synthesis

Our focus is on the interoperability between heterogeneous protocols. By *interoperability* we mean the ability of protocols to *correctly communicate and coordinate* i.e., to *correctly synchronize*. In other words, two systems successfully interoperate if they correctly exchange *compatible conversations* or *compatible traces*. By *heterogeneous protocols* we mean that, although in principle they could interact since they have compatible (i.e., complementary) functionalities, protocols can be characterized by discrepancies that may undermine their ability to seamlessly interoperate (i.e., communicate and coordinate). Discrepancies include incompatible interaction protocols and/or different interfaces meaning different actions and/or data models. Examples of heterogeneous application protocols are the Policeman and Commander of the Patrolling Group of the case study.
In order to enable interoperability among heterogeneous protocols, we devised a theory for the automated synthesis of CONNECTors [3, 4], also called mediating connectors or mediators for short. Figure 5 provides an overview of our methodology.

Our approach takes as input the descriptions of two NSs and in particular their behavioral protocols, described as Labeled Transition Systems (LTSs), together with their ontological information conceptualizing their actions through an application domain ontology. By referring to the case study, the synthesis takes as input the LTS of the Policeman, the LTS of a Commander of the Patrolling Group and of its Guards and their ontologies and follows a process made up by three phases or steps as described in the following.

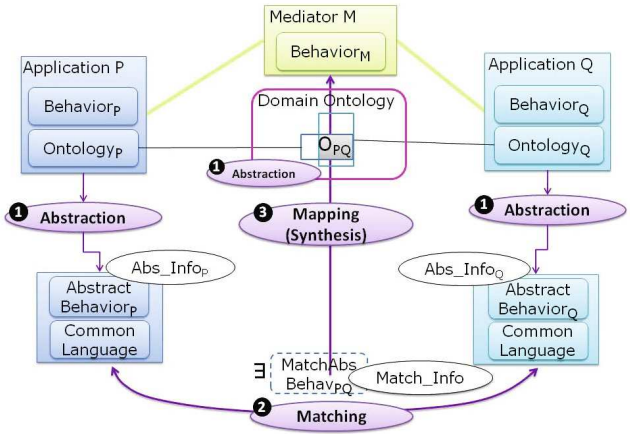


Fig. 5. An overview of the mediator synthesis approach

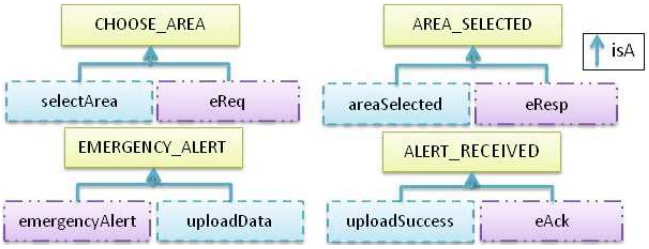


Fig. 6. Domain Ontology (upper case elements) where the ontologies of both protocols have been mapped (Dots and dashes boxes - Ontology of the Guards Control Center; dashed boxes - Ontology of the Policeman)

1. *Abstraction* that makes models comparable and, if possible, reduces their size making it easier and faster to reason on them. Reasoning on ontologies, a common language for both protocols is identified on the domain ontology and used to relabel them. The common language of our case study is illustrated by the elements with upper case names in Figure 6.
2. *Matching* that checks the NSs compatibility identifying possible mismatches. Compatible protocols can potentially interoperate despite they show some differences. That is, communication and coordination between such protocols is possible in principle since they are semantically equivalent and complementary, but cannot be achieved seamlessly because of *heterogeneity*: *mismatches* and/or *third parties conversations*. Examples of mismatches are: protocol languages have (i) different granularity, or (ii) different alphabets; protocols behavior have different sequences of actions with data (i.e., traces) because of (a.1) the order in which actions and data are performed by a protocol is different from the order in which the other protocol performs the complementary actions with data. Protocols behavior may have different sequences of actions also because of (a.2) interleaved actions related to *third parties conversations* i.e., with other systems, the environment. In some cases, as

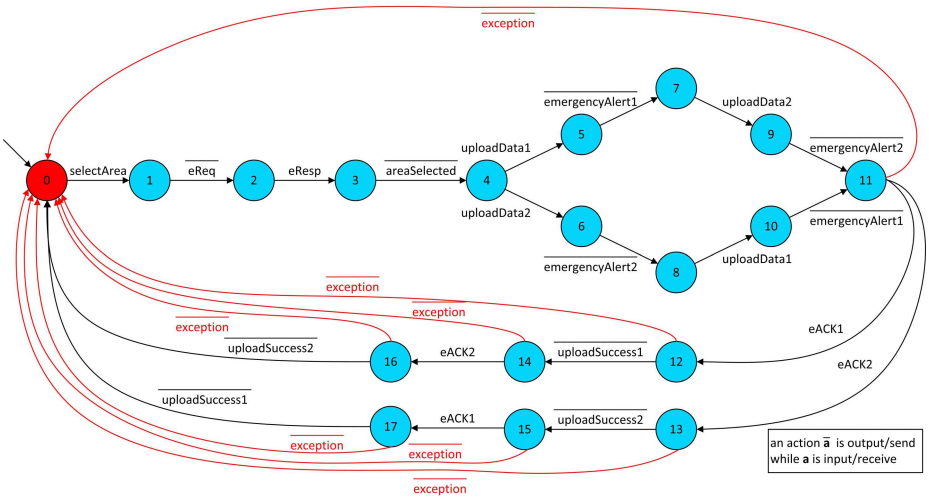


Fig. 7. Synthesised CONNECTor for the case study

for example (i), (ii) and (a.1), it is necessary to properly perform a manipulation of the two languages. In the case (a.2) it is necessary to abstract the third parties conversations that are not relevant to the communication. Referring to the case study, the heterogeneities we identify are what we call signature mismatches [5, 6], i.e., the two protocols use different names for semantically equivalent concepts.
Synchronization between protocols, thus, can be achieved under mediation i.e., through a mediator that while managing such mismatches, allows protocols to effectively exchange compatible traces (sequences of actions).

3. *Mapping* or *Synthesis* that produces a mediator that mediates the found mismatches so enabling the NSs to communicate. Figure 7 illustrates the synthesized CONNECTor between the Policeman and the Commander of the Patrolling Group applications of our case study.
   - The `selectArea` message of the Policeman is translated into an `eReq` message directed to the Commander of the Patrolling Group operating in the area of interest.
   - The `eResp` message of the Commander is translated into an `areaSelected` message for the Policeman.
   - The `uploadData` message of the Policeman is translated into a multicast `emergencyAlert` message to all the Guards (Commander included).
   - Each `eACK` message automatically sent by the Guards' devices that correctly receive the `emergencyAlert` message are translated into a `uploadSuccess` message for the Policeman.

Note that still for figure readability, the illustrated CONNECTor is between one Policeman, a Commander of the Patrolling Group and one Guard.

A **mediator** is then a protocol that allows communication and coordination among compatible protocols by mediating their differences. It serves as the locus where

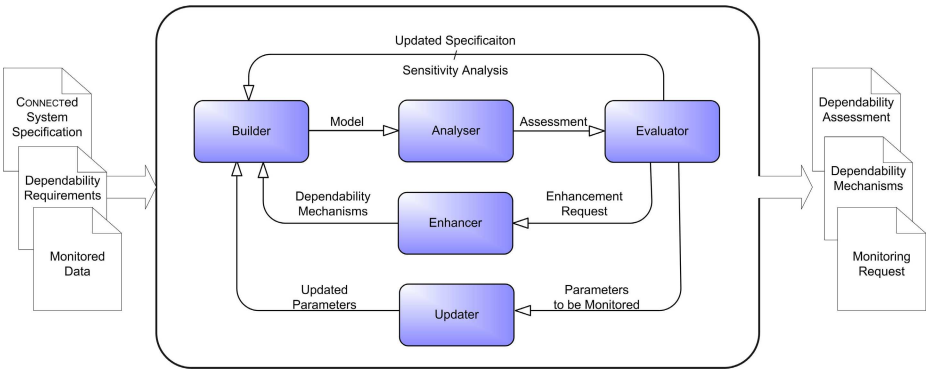


Fig. 8. Architecture of the Dependability&Performance (DEPER) Enabler

semantically equivalent and complementary actions are correctly synchronized thus enabling (a mediated) interoperability among protocols.

In summary, by reasoning about the protocols discrepancies, our theory automatically identifies and synthesizes an *emerging mediator* that solves them thus allowing protocols to interoperate. Automatically synthesized CONNECTors are concrete emergent entities that translate and coordinate mismatching interaction protocols, actions, and/or data models, letting applications interact effectively

5 Pre-deployment Analysis to Support CONNECTor Synthesis

Pre-deployment assessment is a crucial activity to drive the system design towards a realization compliant with the required quality levels. In fact, it allows for early detection of design deficiencies, so as to promptly take the appropriate recovery actions, thus significantly saving in money and time with respect to discovering such problems at later stages. As briefly outlined in the Introduction, the pre-deployment assessment in CONNECT is performed by the Dependability and Performance enabler, introduced in Figure 1 and shortly referred to as DEPER, which exploits State-Based Stochastic modelling and analysis, embedded in an automated process. Figure 8 illustrates the architecture of DEPER that, together with the prototype implementation based on the Möbius evaluation framework, is documented in [7, 8].

Very briefly, the activities of modules Builder, Analyser and Evaluator, triggered in sequence, perform the cycle of the pre-deployment analysis: from the specification of the CONNECTed system (both functional and non-functional) and of the metrics to be analysed, to checking whether the analysis results match with the metrics level, as requested by the NSs.

Evaluator informs Synthesis about the outcome of the check. In case of mismatch, it may receive back a request to evaluate if enhancements can be applied to improve the dependability or performance level of the CONNECTed system, thus calling the intervention of the Enhancer module. In the other case, the CONNECTor's design is considered satisfactory and ready to be deployed, thus terminating the pre-deployment

analysis phase. However, because of possible inaccuracy of model parameters due to potential sources of uncertainty dictated by the dynamic and evolving context, Evaluator also instructs the Updater module about the events to be observed on-line by the Monitor enabler.

The attempts to improve dependability and performance of the CONNECTor consist in extending the model with a dependability mechanism, selected from a library of already defined ones (see [9]), until either a successful mechanism is found, or all the mechanisms are exhausted.

The Updater module provides adaptation of the off-line analysis performed at pre-deployment time to cope with changes in, or inaccurate estimates of, model parameters, through interactions with the Monitor enabler (e.g., because of limited knowledge of the NSs characteristics acquired by Learning/Discovery enablers). It receives from Monitor a continuous flow of data for the parameters under monitoring relative to the different executions of the CONNECTor. Accumulated data are processed through statistical inference techniques. If, for a given parameter, the statistical inference indicates a discrepancy between the on-line observed behaviour and the off-line estimated value used in the model resulting into a significant deviation of the performed analysis, a new analysis is triggered.

5.1 Pre-deployment Analysis in the Terrorist Alert Scenario

Taking as a reference the above described scenario, we focus in the following on the basic interactions between Synthesis and DEPER enablers, performed to exchange requests for dependability analysis of a pre-deployed CONNECTor.

The interaction starts when Synthesis sends a JMS message to DEPER. The message contains the specification of the LTSs of the CONNECTor and of the NSs involved.

Figure 9 depicts the dependability and performance model of the synthesized CONNECTor built by DEPER at design time, using the SAN formalism [10]. The model is obtained through automatic transformation from the LTS specification of the NSs, that is the *SecuredFileSharing* and *EmergencyCall* in the considered scenario. The measure assessed in the evaluation is the *coverage*. Coverage represents a dependability indicator and is given by the percentage of responses the control center receives back from the guards within a certain time T.

Once the message has been received and the SAN models have been built, DEPER starts the coverage analysis through the Möbius tool [11]. The performed analysis is shown in Figure 12 of Section 7 and allows to state that the CONNECTor fully satisfies the coverage requirement with a Timeout greater than 2 time units. Hence, a response is sent, through a JMS message, from DEPER to Synthesis to communicate that the CONNECTor can be dependably deployed.

After the deployment of the CONNECTor a sensitivity analysis from DEPER on the impact of model parameters on the assessment of the selected measure revealed that critical parameters to keep under observation on-line via the Monitor enabler, for the coverage measure, are occurrences of transitions `eACK`. Refining the pre-analysis knowledge on the values assumed for such parameters by real observations constitutes a fundamental step in enhancing the accuracy of the analysis results. In fact, should the initial forecast for these parameters deviate from what is evidenced through repeated executions,

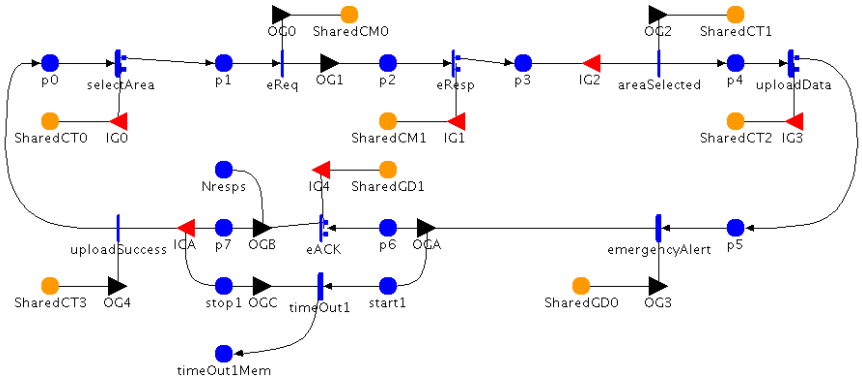


Fig. 9. SAN model of the CONNECTor

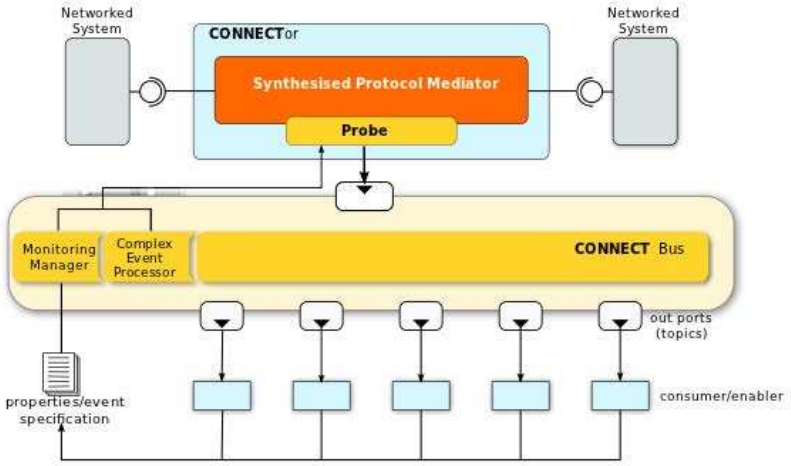


Fig. 10. GLIMPSE architecture

a new analysis round needs to be triggered to understand whether the dependability and performance requirements are still met by the CONNECTed system.

6 Events Observation through Monitoring

Timely and effective run-time adaptation can only be ensured by continuously observing the interactions among the NSs. To this purpose, in CONNECT we have developed a modular, flexible and lightweight monitoring infrastructure, called GLIMPSE[2]. GLIMPSE infrastructure, shown in Figure 10, is totally generic and can be easily applied to different contexts. To provide a better communication decoupling, we adopted a publish-subscribe communication paradigm.

[2] GLIMPSE is an acronym for Generic fLexIble Monitoring based on a Publish-Subscribe infrastructurE.

The lowest level of the monitoring is represented by the probe deployed into the CONNECTor; this probe monitors the messages exchanged among the NSs involved into the communication, possibly applying a local filter in order to decrease the amount of messages sent on the CONNECT bus. Note that such probes are non intrusive data collectors (proxies), i.e., they have no effect on the order and timing of events in the application and do not generate overhead on the communication or on the interacting services.

The second layer of the monitoring infrastructure is represented by the information consumers, the entities interested to obtain the evaluation of a non-functional property or interested to receive notification of occurrences of events/exceptions that may occurs into the CONNECTor.

The gathered information is provided in form of events. An event is an atomic description, a smaller part of a larger and more complex process at application level. In CONNECT, an event represents a method invocation on a remote web service: the invocation, coming from the producer to the consumer, is captured when it comes through the CONNECTor, encapsulated into a specific object, and sent through the CONNECT bus. A Complex Event Processor (CEP) analyzes the atomic events to infer complex events matching the consumer requests, by a rule engine. In the current GLIMPSE implementation, we adopt the Drools Fusion rule language [12] that is open source and can be fully embedded in the realized Java architecture.

Finally, the Manager module is in charge to manage all the communication between the consumers and the CEP.

7 Continuous Run-Time Adaptation

After having performed the pre-deployment analysis phase and the deployment of the CONNECTor, we focus here on its adaptive assessment via the interaction among Synthesis, DEPER and the Monitor enabler. Basically, the dynamicity and evolution of the targeted environment lead to potential sources of uncertainty, which undermine the accuracy of the off-line analysis. To cope with this issue, run-time monitoring is exploited to re-calibrate and enhance the dependability and performance prediction along time. As already pointed out in Section 5, the continuous run-time adaptation of the pre-deployment performed analysis is in charge to the Updater module.

DEPER and Monitor interact by using a Publish/Subscribe protocol. The interaction starts when DEPER sends a JMS message whose payload contains an XML object rule generated using ComplexEventRule classes [7].

Once the CONNECTor is deployed, data (events) derived from real executions are sent by the probe to the CONNECT bus. The Monitor enabler gathers those events and using the CEP component, tries to infer one or more of the patterns to which the DEPER enabler is subscribed.

Upon occurrence of a relevant event the DEPER enabler is notified: the latter, in turn, performs a statistical analysis of the monitored observations and uses such information to check the accuracy of the model analysed before deployment. If the model parameters are found to be inaccurate, DEPER updates the model with the new values,

and performs a new analysis. If the new analysis evidences that the deployed CONNECTor needs adjustments, a new synthesis-analysis cycle starts.

As an example, we consider the steps to refine the accuracy of the *failure probability* of the communication channel between the *EmergencyCall* application and the CONNECTor. We accumulated data generated from several executions of the CONNECTor, in scenario's configurations with a fixed number of 11 guards, which allowed to refine the value of the failure probability from 0.02, assumed during pre-deployment dependability analysis, to 0.1.

Upon updating the parameter, a new dependability analysis is performed in order to verify if the updated CONNECTor still satisfies the requirement. Unfortunately, the analysis shows that coverage measure does not meet the requirement. The CONNECTor needs to be enhanced. Based on the library of dependability mechanism [9], the *retry mechanism* has been automatically selected and implemented in the already developed CONNECTor model, in order to enhance the CONNECTed system and satisfy the requirement.

The retry mechanism consists in re-sending messages that get corrupted or lost during communications, e.g., due to transient failures of communication links. This mechanism is widely adopted in communication protocols, such as TCP/IP for enabling reliable communication over unreliable channels. A typical implementation of the retry mechanism uses time-outs and acknowledgements: after transmitting a message, the sender waits for a message of the receiver that acknowledges successful communication. If the acknowledgement is not received within a certain time interval, the sender assumes that the communication was not successful, and re-transmits the message.

The SAN model of the retry mechanism is shown in Figure 11. On the sender side, the mechanism creates a message re-transmission policy for re-sending the message at most $N = 3$ times; on the receiver side, the mechanism creates a policy for avoiding duplicated reception of messages and for sending acknowledgements. The sender stops re-transmitting the message as soon as it gets an acknowledgement that the message has been successfully received, or after N attempts. A detailed description of the mechanism model can be found in [9].

Figure 12 shows the trend of the *coverage* (on the *y* axis) at increasing values of Timeout (on the x axis). Also, the threshold coverage line as specified in the requirement (set to the value 0.65) is reported. The figure includes three plots, corresponding to: (i) the results of the pre-deployment analysis; (ii) the results of the analysis after the parameters influencing coverage have been updated with actual values from the run-time observations; and (iii) the results of the analysis after both the parameters influencing coverage have been updated and a retry mechanism have been implemented to enhance the CONNECTed system. It is worth noting that coverage value obtained through the pre-deployment analysis is fully satisfying the requirement with timeout value greater than 3 (time units). While the coverage value after updating the failure probability parameter never satisfies the requirement, although the value of timeout increases, which means that the estimation of coverage at pre-deployment time was too optimistic. Finally, we note that the analysis performed considering the actual value of failure probability and including the enhanced mechanism provides results on coverage that satisfies the coverage requirement.

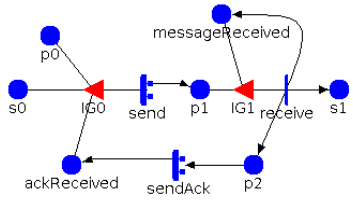


Fig. 11. Retry mechanism

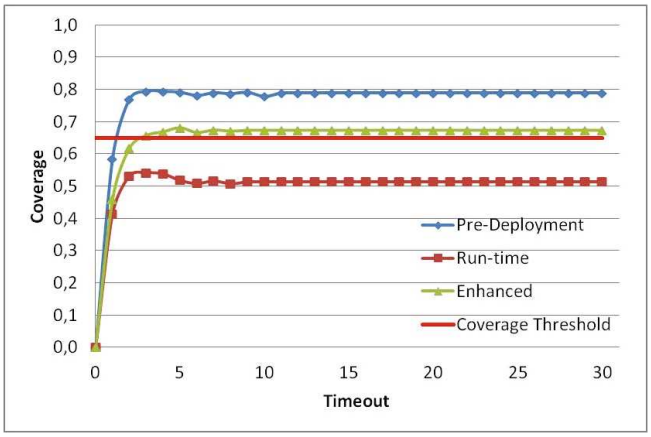


Fig. 12. Trend of Coverage as a function of Timeout

The CONNECTor needs to be enhanced; therefore DEPER informs the Synthesis enabler about the analysis results and appropriate actions are taken by Synthesis (typically, a new CONNECTor is synthesised).

8 Related Work

The automated synthesis of application-layer CONNECTors relates to a wide number of works in the literature within different research areas.

The *Ubiquitous Computing* (UbiComp) proposed by Weiser [13] has a key principle that is to make the computer able to vanish in the background to increase their use making it in an efficient and invisible manner to users. Our CONNECTors fit perfectly the ubiquitous vision where each NS maintains its own characteristics, being able to communicate and cooperate with the others without having any prior knowledge of them thanks to the support provided by the mediators that masks divergencies making them appear homogeneous.

Interoperability and *mediation* have been investigated in several contexts, among which protocol conversion [14–16], integration of heterogeneous data sources [17], software architecture [18], architectural patterns [19], design patterns [20], patterns of connectors [21, 22], Web services [23–27], and algebra to solve mismatches [28] to mention a few.

A lot of work has also been devoted to *connectors* like for example [29–33] to mention few. A work strictly related to the mediators is the seminal work by Yellin and Strom [34]. With respect to our synthesis approach, this work prevents to deal with ordering mismatches and different granularity of the languages (see [5, 6] for a detailed mismatches description). Other works related to our but posing the focus on different problems are [35] and [36].

Stochastic model-based approaches for quantitative analysis of non-functional properties have been largely developed along the last decades and documented in a huge literary production on this topic. The already cited papers [1, 37] provide a survey of the most popular ones. The choice of the most appropriate type of model to employ depends upon the complexity of the system under analysis, the specific aspects to be studied, the attributes to be evaluated, the accuracy required, and the resources available for the study. The prototype implementation of our DEPER enabler is based on Stochastic Activity Networks (SANs) [10], a variant of the Stochastic Petri Nets class.

With regard to monitoring, various approaches have been recently proposed. Similarly to GLIMPSE, also [38] presents an extended event-based middleware with complex event processing capabilities on distributed systems, adopting a publish/subscribe infrastructure, but it is mainly focused on the definition of a complex-event specification language. The aim of GLIMPSE is to give a more general and flexible monitoring infrastructure for achieving a better interpretability with many possible heterogeneous systems. Another monitoring architecture for distributed systems management is presented in [39]. Differently from GLIMPSE, this architecture employs a hierarchical and layered event filtering approach. The goal of the authors is to improve monitoring scalability and performance for large-scale distributed systems, minimizing the monitoring intrusiveness.

A prominent part of our framework is in the combined usage of pre-deployment model-based analysis and run-time observations via monitoring. Preliminary studies that attempt combining off-line with on-line analysis have already appeared in the literature. A major area on which such approaches have been based is that of autonomic computing. Among such studies, in [40], an approach is proposed for autonomic systems, which combines analytic availability models and monitoring. The analytic model provides the behavioural abstraction of components/subsystems and of their interconnections and dependencies, while statistical inference is applied on the data from real time monitoring of those components and subsystems, to assess parameter values of the system availability model. In [41], an approach is proposed to carry out run-time reliability estimation, based on a preliminary modelling phase followed by a refinement phase, where real operational data are used to overcome potential errors due to model simplifications. Our approach aims at proposing a general and powerful evaluation framework, tailored to a variety of dependability and performance metrics, to meet a wide spectrum of system requirements and adaptation needs.

9 Conclusions

We have introduced the ambitious vision of the CONNECT project for an eternally and dependably CONNECTed world. Of the complex CONNECT architecture under development, we have focused here on the Synthesis enabler, which derives on-the-fly a

mediator enabling the functional interoperation among heterogenous NSs; the Dependability&Performance enabler, which applies Stochastic model-based analysis for assessing the desired non-functional properties; and the Monitor, which observes the run-time CONNECTor behaviour. We have discussed on a case study their integrated usage to allow for adaptive analysis accounting for possible inaccurate information or potential evolution of the involved NSs. We refer to a library of adaptation patterns that DEPER suggests to Synthesis to enhance the CONNECTor and make it compliant with the expected non-functional properties. At present, Synthesis uses such suggestion to synthesize a new CONNECTor that can satisfy the non-functional requirements. In future, we will investigate approaches for on-the-fly adaptation of the CONNECTor, where possible.

For reasons of space, we could not cover other important enablers in the CONNECT architecture. Further information can be obtained from the project web site.

Acknowledgements. This work has been partially supported by the European Project CONNECT Grant Agreement No.231167.

References

1. Bondavalli, A., Chiaradonna, S., Giandomenico, F.D.: Model-based evaluation as a support to the design of dependable systems. In: Diab, H.B., Zomaya, A.Y. (eds.) Dependable Computing Systems: Paradigms, Performance Issues, and Applications, pp. 57–86. Wiley (2005)
2. CONNECT Consortium: Deliverable 6.1 – Experiment scenarios, prototypes and report – Iteration 1 (2011)
3. Inverardi, P., Issarny, V., Spalazzese, R.: A Theory of Mediators for Eternal Connectors. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010, Part II. LNCS, vol. 6416, pp. 236–250. Springer, Heidelberg (2010)
4. Spalazzese, R., Inverardi, P., Issarny, V.: Towards a formalization of mediating connectors for on the fly interoperability. In: Proceedings of the WICSA/ECSA 2009, pp. 345–348 (2009)
5. Spalazzese, R., Inverardi, P.: Mediating Connector Patterns for Components Interoperability. In: Babar, M.A., Gorton, I. (eds.) ECSA 2010. LNCS, vol. 6285, pp. 335–343. Springer, Heidelberg (2010)
6. Spalazzese, R., Inverardi, P.: Components interoperability through mediating connector pattern. In: WCSI 2010, arXiv:1010.2337. EPTCS, vol. 37, pp. 27–41 (2010)
7. Bertolino, A., Calabró, A., Di Giandomenico, F., Nostro, N.: Dependability and Performance Assessment of Dynamic CONNECTed Systems. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 350–392. Springer, Heidelberg (2011)
8. Masci, P., Martinucci, M., Di Giandomenico, F.: Towards automated dependability analysis of dynamically connected systems. In: Proc. IEEE International Symposium on Autonomous Decentralized Systems, Kobe, Japan, pp. 139–146. IEEE (June 2011)
9. Masci, P., Nostro, N., Di Giandomenico, F.: On Enabling Dependability Assurance in Heterogeneous Networks through Automated Model-Based Analysis. In: Troubitsyna, E.A. (ed.) SERENE 2011. LNCS, vol. 6968, pp. 78–92. Springer, Heidelberg (2011)
10. Sanders, W.H., Malhis, L.M.: Dependability evaluation using composed SAN-based reward models. Journal of Parallel and Distributed Computing 15, 238–254 (1992)
11. Daly, D., Deavours, D.D., Doyle, J.M., Webster, P.G., Sanders, W.H.: Möbius: An Extensible Tool for Performance and Dependability Modeling. In: Haverkort, B.R., Bohnenkamp, H.C., Smith, C.U. (eds.) TOOLS 2000. LNCS, vol. 1786, pp. 332–336. Springer, Heidelberg (2000)

12. Drools fusion: Complex event processor, http://www.jboss.org/drools/drools-fusion.html
13. Weiser, M.: Hot Topics: Ubiquitous Computing. IEEE Computer (1993)
14. Calvert, K.L., Lam, S.S.: Formal methods for protocol conversion. IEEE Journal on Selected Areas in Communications 8, 127–142 (1990)
15. Lam, S.S.: Correction to "protocol conversion". IEEE Trans. Software Eng. 14, 1376 (1988)
16. Okumura, K.: A formal protocol conversion method. In: SIGCOMM, pp. 30–37 (1986)
17. Wiederhold, G.: Mediators in the architecture of future information systems. IEEE Computer 25, 38–49 (1992)
18. Garlan, D., Shaw, M.: An introduction to software architecture. Technical Report CMU-CS-94-166, Carnegie Mellon University (1994)
19. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture. A System of Patterns, vol. 1. Wiley, Chichester (1996)
20. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Resusable Object-Oriented Software. Addison-Wesley Professional (1995)
21. Wermelinger, M., Fiadeiro, J.L.: Connectors for mobile programs. IEEE Trans. Softw. Eng. 24, 331–341 (1998)
22. Spitznagel, B.: Compositional Transformation of Software Connectors. PhD thesis, Carnegie Mellon University (2004)
23. Motahari Nezhad, H.R., Xu, G.Y., Benatallah, B.: Protocol-aware matching of web service interfaces for adapter development. In: Proceedings of the 19th International Conference on World Wide Web, WWW 2010, pp. 731–740. ACM, New York (2010)
24. Cimpian, E., Mocan, A.: WSMX Process Mediation Based on Choreographies. In: Bussler, C.J., Haller, A. (eds.) BPM 2005. LNCS, vol. 3812, pp. 130–143. Springer, Heidelberg (2006)
25. Vaculín, R., Sycara, K.: Towards automatic mediation of OWL-S process models. In: IEEE International Conference on Web Services, pp. 1032–1039 (2007)
26. Williams, S.K., Battle, S.A., Cuadrado, J.E.: Protocol Mediation for Adaptation in Semantic Web Services. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 635–649. Springer, Heidelberg (2006)
27. Cavallaro, L., Di Nitto, E., Pradella, M.: An Automatic Approach to Enable Replacement of Conversational Services. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 159–174. Springer, Heidelberg (2009)
28. Dumas, M., Spork, M., Wang, K.: Adapt or Perish: Algebra and Visual Notation for Service Interface Adaptation. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 65–80. Springer, Heidelberg (2006)
29. Spitznagel, B., Garlan, D.: A compositional formalization of connector wrappers. In: ICSE, pp. 374–384 (2003)
30. Fiadeiro, J.L., Lopes, A., Wermelinger, M.: Theory and practice of software architectures. Tutorial at the 16th IEEE Conference on Automated Software Engineering, San Diego, CA, USA, November 26-29 (2001)
31. Lopes, A., Wermelinger, M., Fiadeiro, J.L.: Higher-order architectural connectors. ACM Trans. Softw. Eng. Methodol. 12, 64–104 (2003)
32. Barbosa, M.A., Barbosa, L.S.: Specifying Software Connectors. In: Liu, Z., Araki, K. (eds.) ICTAC 2004. LNCS, vol. 3407, pp. 52–67. Springer, Heidelberg (2005)
33. Bruni, R., Lanese, I., Montanari, U.: A basic algebra of stateless connectors. Theor. Comput. Sci. 366, 98–120 (2006)
34. Yellin, D.M., Strom, R.E.: Protocol specifications and component adaptors. ACM Trans. Program. Lang. Syst. 19 (1997)
35. Tivoli, M., Inverardi, P.: Failure-free coordinators synthesis for component-based architectures. Sci. Comput. Program. 71, 181–212 (2008)

36. Canal, C., Poizat, P., Salaün, G.: Model-based adaptation of behavioral mismatching components. IEEE Trans. Software Eng. 34, 546–563 (2008)
37. Nicol, D.M., Sanders, W.H., Trivedi, K.S.: Model-based evaluation: from dependability to security. IEEE Transactions on Dependable and Secure Computing 1, 48–65 (2004)
38. Pietzuch, P., Shand, B., Bacon, J.: Composite event detection as a generic middleware extension. IEEE Network 18, 44–55 (2004)
39. Hussein, E.A.S., Abdel-wahab, H., Maly, K.: HiFi: A New Monitoring Architecture for Distributed Systems Management. In: Proceedings of ICDCS, pp. 171–178 (1999)
40. Mishra, K., Trivedi, K.S.: Model Based Approach for Autonomic Availability Management. In: Penkler, D., Reitenspiess, M., Tam, F. (eds.) ISAS 2006. LNCS, vol. 4328, pp. 1–16. Springer, Heidelberg (2006)
41. Pietrantuono, R., Russo, S., Trivedi, K.S.: Online monitoring of software system reliability. In: Proc. EDCC 2010 - 2010 European Dependable Computing Conference, pp. 209–218. IEEE Computer Society (2010)

Enhanced Connectors Synthesis to address Functional, Performance, and Dependability aspects

Nicola Nostro

University of Florence

Romina Spalazzese

University of L'Aquila, L'Aquila, Italy

Felicita Di Giandomenico

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", CNR, Pisa Italy

Paola Inverardi

University of L'Aquila, L'Aquila, Italy

Abstract

Our everyday life is pervaded by the use of a number of heterogeneous systems that are continuously and dynamically available to interoperate in the networked environment. The evolving nature of this environment with no a-priori knowledge of the systems, requires automated solutions as means to achieve interoperability with the needed level of flexibility. We already investigated and proposed an approach to the automated synthesis of CONNECTors (or mediators) between heterogeneous Networked Systems (NSs) for their *functional interoperability* at application layer.

In this paper we propose (i) an *approach* to *enhance* the CONNECT*ors* taking into account performance and dependability aspects and (ii) a CONNECT*or adaptation process*, related to the performance and dependability mechanisms, spanning pre-deployment time and run-time, and (iii) a stochastic model-based implementation of the performance and dependability analysis. By reasoning on systems' specification, during the *pre-deployment phase* the approach produces a mediator that satisfies the functional, performance and dependability requirements. At *run-time*, if a performance or dependability violation occurs, by reasoning on the new specification, the approach

This paper is under refinement for submission to an international, peer-reviewed journal – December 10, 2012

identify the proper mechanism to solve the problem and update the CONNECTor accordingly.

Key words: Connector synthesis, Dependability, Performance, Inteoperability

1. Introduction

An always increasing number of heterogeneous systems pervade our everyday life. Nowadays, we use more and more systems that are dynamically available in the networked environment and that, by interoperating with other systems, allow us to reach some goal. The goal can be about both functional and/or non functional aspects and have to be satisfied in order for two systems to interoperate. Abstractly some of these heterogeneous applications could interact, since they have compatible functionalities and similar interaction protocols. Nevertheless, their ability to seamlessly interoperate may be undermined by some resolvable mismatch in their protocols (e.g., interactions order or input/output data formats) and non functional requirements. Solving such mismatches and meeting the non functional requirements, asks for applications' adaptation through a CONNECTor. Further, in this evolving environment, there is no a-priori knowledge of the systems until they are discovered and possibly learned and automated solutions appear to be the only way to enable composition and interoperability of applications with the needed level of flexibility.

The described context is considered by the CONNECT European project[1] whose aim is to allow seamless interoperability between heterogeneous protocols at various levels. The project adopted as solution an approach for the on the fly synthesis of *emergent* CONNECTors via which Networked Systems communicate. The emergent CONNECTors (or mediators) are system entities synthesized according to the behavioral semantics of protocols run by the interacting parties at application and middleware layers. The synthesis process is based on a formal foundation for CONNECTors, which allows learning, reasoning about and adapting the interaction behavior of NSs at run-time through CONNECTors. We already investigated and proposed an approach to the automated synthesis of CONNECTors (or mediators) between heterogeneous NSs for their *functional interoperability* at application layer [16],

[1] CONNECT Web Site - `http://connect-forever.eu/`

[17]. However, effective interoperability also requires that the CONNECT*ed system*, i.e., networked systems and CONNECTor, provides *non functional interoperability*, i.e., the required non-functional properties, during their interoperation. Thus, a suitable and adaptive framework is required that provides a solution to both functional and non functional interoperability.

In this paper we propose (i) an approach to *enhance* the functional CONNECT*ors* taking into account performance and dependability aspects, (ii) a CONNECT*or adaptation process*, spanning pre-deployment time and run-time to preserve the CONNECTor adequacy with respect to non functional requirements along time, and (iii) a stochastic model-based implementation of the performance and dependability analysis. By reasoning on systems' specification, during the *pre-deployment phase* the approach produces a mediator that does not satisfies the functional, performance and dependability requirements. At *run-time*, when a performance or dependability violation occurs, the approach by reasoning on the new specification, identifies the proper mechanism to solve the problem and properly update the CONNECTor.

Regarding the synthesis of CONNECTors meeting both functional and performance concerns, we already conducted investigations and an approach is presented in [14]. Such approach is complementary with respect to the work presented in this paper because tackles different aspects. One can take advantage from the synergic application of both of them to reinforce the CONNECTor with respect to the non functional requirements. Indeed, the work in [14] takes place at pre-deployment time, when heterogeneous NSs requesting interoperability with some performance requirement, trigger the need of a CONNECTor. Its aim is to produce a CONNECTor satisfying both the functional and performance required characteristics. Instead, this work take place both at pre-deployment time and run-time and its aim is to cope with problems arising from the execution environment due to the uncertainties about the knowledge of the environment itself at pre-deployment time and the evolution of the working context. These different perspectives make both approaches relevant and strategic.

The remainder of the paper is organized as follows. Section 2 describes the context we consider, the process followed by our approach, and background notions. Section 3 illustrates a case study that is used for both explanation and experimentation purposes. Then, Section 4 provides a detailed description of our approach and Section 5 illustrates related work. Finally, Section 6 concludes the paper.

2. Setting the Context

A number of heterogeneous networked systems, e.g., tablet, desktop, smartphone, laptop, and robots, are dynamically available in the networked environment. NSs heterogeneity spans many aspects and we focus on their *application layer*. In the following we describe an example of heterogeneous applications that will be detailed and extended in Section 3. The laptop in Figure 1 owns and runs a *client application* represented by the gray icon *Appl1* that is suited to directly interoperate with a server application represented by the gray icon *Appl1*, owned by the Unmanned Ground Vehicle (UGV), with some performance and dependability requirements.

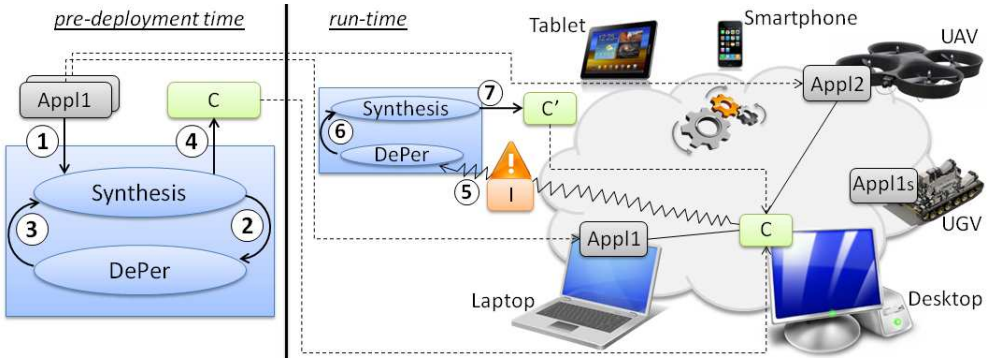


Figure 1: An Overview

Instead, as it is, *Appl1* is not able to directly interact with the *server application* represented by the gray icon *Appl2* owned by and running on the Unmanned Aerial Vehicle (UAV) nevertheless in principle *Appl1* and *Appl2* are compatible. However, due to some protocol discrepancies and non functional concerns, they cannot seamlessly interoperate and a CONNECTor C that mediates their differences is needed in between.

In CONNECT, we consider that the *NSs and their applications are black box* and that for any application they expose in their interface: the interaction behavior description, the non functional properties description, and possibly the non functional requirements on potential interactions with other NSs. Thus, *the* CONNECT*or is the only locus where we can act to make the* CONNECT*ed system satisfy both functional and non functional interoperability.* From a functional point of view, we build a CONNECTor such that makes the NSs behaviors compatible through a proper interaction with them. For

instance, if a NS sends data with a finer granularity with respect to another NS, the mediator has to first collect all the pieces of data information and then send them properly to the other. From a non functional perspective, to make the CONNECTed system satisfy the requirements, we can act on the CONNECTor to suitably enrich it. For instance, in the described scenario, performance and dependability concerns arise because *Appl1* wants to interoperate with *Appl2* with some performance and dependability constraints as for example: the latency between a command sent by Appl1 and the reception of the corresponding acknowledgment sent by Appl2 must be under a threshold.

In the following we overview our *approach* to synthesize a CONNECTor, that meet both functional and some non functional concerns, including also the description of our *adaptation process*. A preliminary study has been conducted and described in [3] where a cycle involving the Synthesis, DEPER and Monitor Enablers[2] has been identified as high level approach. In the following we mainly focus on the collaboration between DEPER and Synthesis Enablers and refine the vision provided in [3] by detailing the Synthesis-DEPER cooperation.

Our approach. It takes place both at *pre-deployment time* and *run-time*, and its aim is to cope with problems arising from the execution environment due to the uncertainties about the knowledge of the environment itself at pre-deployment time and the evolution of the working context.

During the *pre-deployment time*, our approach takes as input applications' specification (see ① in Figure 1). By reasoning on them, a mediator is automatically synthesized solving discrepancies and enabling the functional interoperation among them (Synthesis). By taking as input the synthesized mediator (see ② in Figure 1), stochastic model-based analysis assesses the desired non functional properties, and a-priori feedback is provided to Synthesis (see ③ in Figure 1) about how the system is expected to operate and how to possibly enrich the previously synthesized mediator (DEPER). We assume that NSs are able to interoperate with the mechanisms we use to enrich the CONNECTor. The output of this collaborative computation at

[2]In the CONNECT project an Enabler is intuitively a networked entity that incorporate the intelligence and logic offered by CONNECT. To enable a required connection it is needed the collaboration of several enablers.

pre-deployment time, given some initial systems knowledge, is a CONNECTor C satisfying functional, performance, and dependability requirements (see ④ in Figure 1).

At *run-time* the applications and the synthesized mediator, also equipped with probes to monitor the connected system [3], are deployed and running on some devices. When a performance or dependability violation occurs (see ⑤ in Figure 1), it is identified by the probes and the **adaptation process** is triggered. By reasoning on the new systems specification, that is the input systems specification modified through their run-time observation, DEPER identifies a proper mechanism to solve, if possible, the problem/violation choosing among *retry, majority voting, probing, error correction* [22]. Subsequently, DEPER triggers the Synthesis by providing it the needed information (see ⑥ in Figure 1) to properly *enrich* the previously synthesized CONNECTor with the identified mechanism. The output of this collaborative computation at run-time, given some performance or dependability violation occurrence, is a new CONNECTor satisfying the functional, performance, and dependability requirements (see ⑦ in Figure 1). This concludes the adaptation process cycle. It is worth to notice that the run-time cycle of our approach is repeated each time a new violation is detected, while the pre-deployment time activities are done only once.

Background Model. In the following, we recall notations inherited by the CONNECT project to describe the NSs.

Networked Systems' applications specification. We use Labeled Transition System (LTS) to model applications' protocol and refer to ontologies to conceptualize their actions and input/output data, and to reason on them.

Specifically, we consider what we call *enhanced Labeled Transition Systems (eLTS)* that is a quintuple (S, L, D, F, s_0) where: S is a finite non-empty set of states; L is a finite set of labels describing actions with data; $D \subseteq S \times L \times S$ is a transition relation; $F \subseteq S$ is the set of final states; $s_0 \in S$ is the initial state. The eLTS' labels are of the form $<op,In,Out>$ where: *op* is an observable action referring to an ontology concept or is an internal action denoted by τ; an action can have output/input direction denoted by an overbar or no overbar on the action respectively, e.g. $\overline{act}$ or act. In, Out are the sets of input/output data that can be produced/expected whose elements refer to ontology elements. We are able to describe the following actions with data: (1) *output action with outgoing parameters and incoming return data* $<\overline{op}, In, Out>$ where In is produced while Out is expected; (2)

input action with incoming parameters and outgoing return data <*op*, *In*, *Out*> where *In* is expected while *Out* is produced. One at a time *In* or *Out* might be empty because no input/output data is expected/produced. This leads to 4 variants of the actions, two for (1) and two for (2). Note that (1) (<$\overline{op}$,*In*,*Out*>) can be equivalently described by the two following action primitives: <$\overline{op}$,*In*,−> and <$\overline{op}$,−,*Out*>. This applies similarly to (2) (<*op*, *In*, *Out*> can be described as <*op*,*In*,−> and <*op*,−,*Out*>). Between protocols, we assume synchronous communications on complementary actions. Actions <op_1,In_1,Out_1> and <op_2,In_2,Out_2> are complementary iff $op_1 = \overline{act}$ and $op_2 = act$ and $In_2 \subseteq In_1$ and $Out_1 \subseteq Out_2$ (or similarly with exchanged roles of op_1 and op_2). Moreover, we consider finite traces by assuming a bound on the number of cycles execution.

Ontologies describe domain-specific knowledge through concepts and relations, e.g. the subsumption: a concept C *is subsumed by* a concept D in a given ontology $\mathcal{O}$, noted by $C \sqsubseteq D$, if in every model of O the set denoted by C is a subset of the set denoted by D [1]. We assume that each NS action and datum refer to some concept of an existing domain ontology so that we can reason on them in order to find a common language between protocols.

Non functional concerns specification: properties, requirements, mechanisms. The NSs dependability and performance model and the dependability and performance properties required by the NSs are expressed as metrics and guarantees.

Metrics are arithmetic expressions that describe how to obtain a quantitative assessment of the properties of interest of the CONNECTed system. They are expressed in terms of transitions and states of the eLTS specification of the NSs. Currently we are using the XML notation, to describe the metrics.

Guarantees are boolean expressions that are required to be satisfied on the metrics.

In order to enrich the CONNECTor, the four *mechanisms* that we can leverage on and apply singly or in combination are: *retry, majority voting, probing, and error correction.*

The Retry, Majority Voting and Error Correction mechanisms are specifically applied if the metric to be analysed is related to dependability aspects. These mechanisms can be applied when the metric under analysis is a function of failure probabilities of all the communications/actions between the CONNECTor and the NSs and there are no timing constraints on the application.

The Probing mechanism is applied when the metric to be analyzed is related to performance aspects instead. This mechanism is applied if there are generic constraints, in particular related to timing aspects. In general the mechanism allows to exploit the most performable communication channels available for transmission, selecting for the use the most efficient one.

In the following we briefly describe the application of the four mechanisms to the CONNECTor.

Retry mechanism. The CONNECTor sends again its request n times if a confirmation/ACK is not received back from a NS. The messages sent must have a sequence ID in order to identify them.

Majority voting mechanism. A portion of the CONNECTor needs to run on each NS. On the one hand, the CONNECTor needs to access the data in order to send it on several channels. On the other hand, the CONNECTor has to choose the correct data to be passed to the NS based on a voting policy.

Probing mechanism. A portion of the CONNECTor needs to run on each NS. On the one hand the CONNECTor checks the performance of redundant channels in order to send the data over the better one. On the other hand, the CONNECTor has to receive from either channels and pass the data to the NS.

Error correction mechanism. A portion of the CONNECTor needs to run on each NS. On the one hand the CONNECTor needs to access the data to be sent and to know some additional information necessary for the error detection and reconstruction (i.e., correction) in order to send them over two channels. On the other hand, the connector has to receive the data and the additional information from two channels and correct the data in case of error thus passing the correct data to the NS.

3. GMES Case Study: the Forest-fire Emergency

In this section we present our scenario, based on the Global Monitoring for Environment and Security (GMES) European Programme[3], in order to show how Synthesis and DEPER work in an integrated way. The GMES emergency management service covers different catastrophic circumstances, e.g., floods, earthquakes, volcanic eruptions.

Scenario. We concentrate on the management of a forest fire emergency

[3]http://www.gmes.info/

situation [11] close to a border village and a factory between Country A and B. The scenario is illustrated in Figure 2.

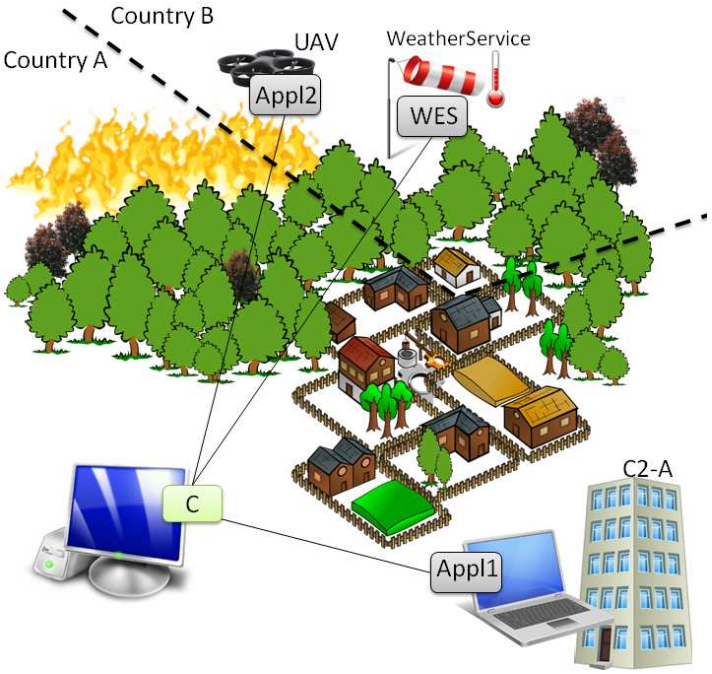


Figure 2: The Forest-fire Emergency case study

Country A's Command and Control fire operations center (C2-A) is in charge of forest monitoring and forest fire management. During a forest fire, the fire goes wider and there is a direct threat to the village and the factory. Thus, Country A asks Country B to give it support. Country B provides reinforcement resources that once deployed are available to be used by the Command and Control Center of Country A. The resources provided by Country B have the same aim with respect to similar resources belonging to Country A, (e.g., to provide high quality images or weather information of the area interested by the fire), but use different protocols. Thus, it is needed to synthesize a mediator to allow Country A to exploit them during the emergency.

The applications we consider in the scenario are: a) the Command and Control Center application of Country A -*Appl1*, b) a fleet of Unmanned Aerial Vehicles (UAVs) applications of Country B -*Appl2*- each equipped with various Video Cameras to get a better view of the fire front close to the

village, and c) the Weather Service application of Country B - *WES*- in order to continuously get information about temperature, humidity and wind of the area interested by the fire.

Country B provides to Country A applications that are compatible with a), b), and c) in principle. However, due to some protocol discrepancies and non functional concerns, they cannot seamlessly interoperate and a CONNECTor that mediates their differences is needed in between.

In the described scenario, a *performance* concern arises because when the Command and Control Center sends an order to move to a robot (e.g., UAV) it wants to receive an acknowledgment within a certain time (*latency*). The latency requirement specified by C2-A is *to receive an acknowledgement within 5 time units*. A *dependability* issue, instead, arises when considering the Weather Service and concerns the percentage of weather data the Command and Control Center correctly receives from the Weather Service (*coverage*). The coverage requirement specified by the C2-A is *to correctly receive at least the 90% of the required weather data*. In order for the connected system to meet the above mentioned non functional requirements, a proper CONNECTor is needed.

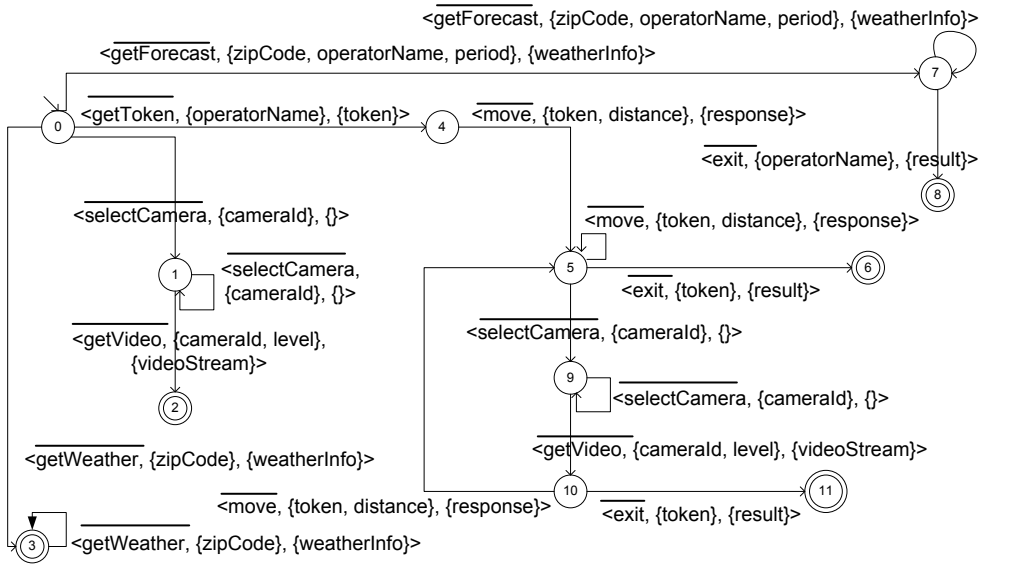
In the following we describe the behaviour of NSs involved in our scenario.

3.1. Command and Control Center of Country A

The Command and Control Center (C2-A) Networked System interacts with two resources: the Weather Service and the UAV of Country B leveraging on protocols used to interact with similar resources belonging to Country A. Figure 3 shows the LTS of C2-A where both interactions, with the UAV and the WS, are represented.

To get weather information about an area, C2-A simply sends a *getWeather* message with the *zipCode* of the area and receives back *weatherInfo* from the Weather Station (Service) including the temperature, humidity and wind conditions. C2-A can also get a weather forecast on the area of interest in a specified period by sending a *getForecast*.

To access the resources operating in the field and equipped with one or more video cameras, e.g., the UGVs or UAVs, C2-A first needs to authenticate with the resource sending a *getToken* message and receiving back the *token*. Then, it can instruct the resource to move *forward*, *backward*, *left* or *right*, by sending the proper message and receiving back a *response* message about the movement. Then, C2-A can choose a camera (*selectCamera*



The action *move* (<move, {token, distance}, {response}>) represents four alternative commands to move: *backward, forward, left, right* (<backward, {token, distance}, {response}>, <forward, {token, distance}, {response}> , <left, {token, distance}, {response}>, <right, {token, distance}, {response}> respectively).

Figure 3: LTS of the Command and Control Center

installed on the resource and receive the video stream (*getVideo*) with a specified zoom *level*.

3.2. Weather Service of Country B

Figure 4 shows the eLTS of the Weather Service (WS). The WS expects to receive either a weather forecast request (*getForecast*) about a specified area of interest (*zipCode*) in a certain *period* of time, or the current weather information (*getTemperature*, *getHumidity*, *getWind*). In the case study the user is interested to know the current weather information.

3.3. UAV and integrated Video Cameras of Country B

Figure 5 shows the eLTS representing the UAV and its integrated Video Cameras. The UAV first expects to receive an identification request (*getIdentifier*) for which it gives back an *identifier* followed by a *takeOff* request which result is notified through a *response*.

After the *takeOff*, the UAV expects to receive either a request to *land* and then to *quit* or to *move* (left, right, forward, backward, up or down). For each

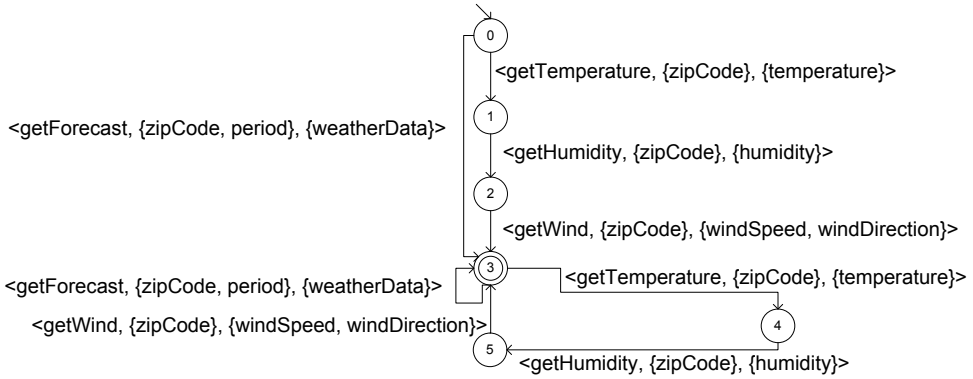
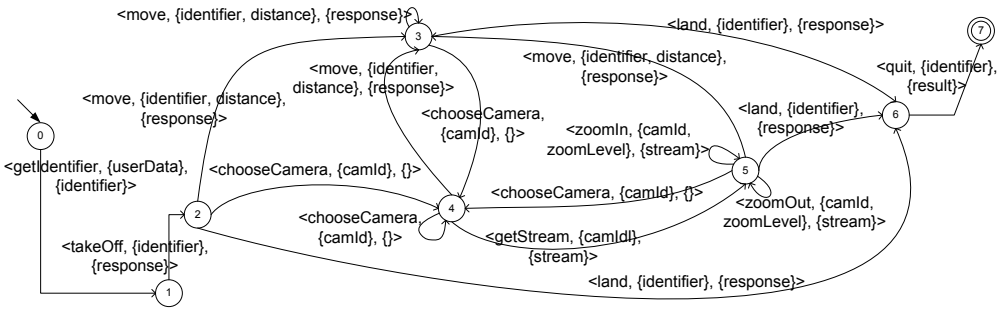


Figure 4: Behaviour of the Weather Service



The action *move (*<move, {identifier, distance}, {response}>) represents four alternative commands to move: *backward, forward, left, right* (<backward, {identifier, distance}, {response}>, <forward, {identifier, distance}, {response}> , <left, {identifier, distance}, {response}>, <right, {identifier, distance}, {response}> respectively).

Figure 5: Behaviour of the UAV and its integrated Video Cameras

movement order the UAV sends back a *response* message. Moreover, at any time while moving during the flight, the UAV can receive a *chooseCamera* request and send back to the requester the real time video stream of the chosen video camera perform. The *chooseCamera* can be followed by any number of *zoomIn* and/or *zoomOut* requests until the reception of a request to land and then to quit.

4. Our Approach to the Enhanced Connector Synthesis

In the following we detail our approach for the synthesis of enhanced CONNECTors to address functional, performance, and dependability aspects

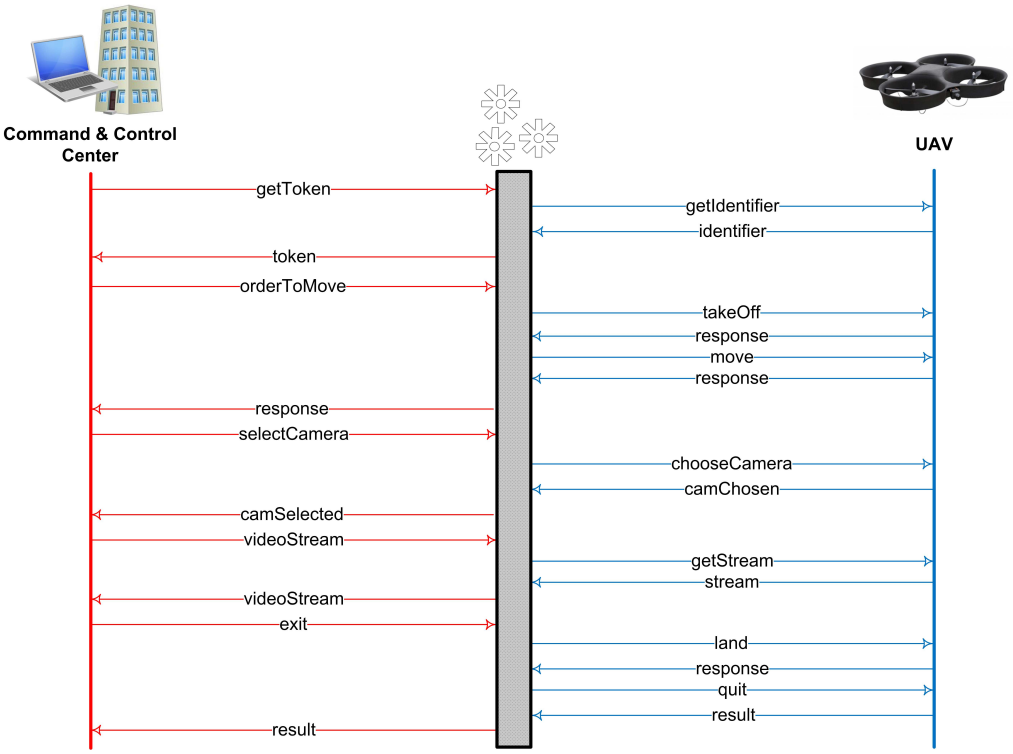


Figure 6: An interaction between C2-A and UAV

leveraging on the forest fire case study. Our approach takes place both at *pre-deployment time* and *run-time*, and its aim is to cope with problems arising from the execution environment due to the uncertainties about the knowledge of the environment itself at pre-deployment time and the evolution of the working context.
We first investigated a scenario considering black box NSs and then, by relaxing this assumption, we considered another scenario where NSs trust and authorize the CONNECTor to access them in order to apply some mechanisms that enhance the synthesized CONNECTor.

4.1. Background: Synthesis and DEPER

-*Synthesis*- In [16] and [17] it has been proposed an approach for the automated synthesis of mediators that overcomes interoperability problems between two heterogeneous emerging protocols, given the NSs models, ontology describing the domain-specific knowledge -and a bound on the number

C2 control center	UAV	WEATHER SERVICE
<getToken, {operatorName}, {token}>	<getIdentifier, {userData}, {identifier}>	---
<right, {token, distance}, {response}>	<moveRight, {identifier, distance}, {response}>	---
<left, {token, distance}, {response}>	<moveLeft, {identifier, distance}, {response}>	---
<forward, {token, distance}, {response}>	<moveForward, {identifier, distance}, {response}>	---
<backward, {token, distance}, {response}>	<moveBackward, {identifier, distance}, {response}>	---
<selectCamera, {cameraId}, {}>	<chooseCamera, {camId}, {}>	---
<getVideo, {cameraId, level}, {videoStream}>	<getStream, {camId}, {stream}>	---
<zoomIn, {cameraId, level}, {videoStream}>	<zoomIn, {camId, zoomLevel}, {stream}>	---
<zoomOut, {cameraId, level}, {videoStream}>	<zoomOut, {camId, zoomLevel}, {stream}>	---
---	<land, {identifier}, {response}>	---
---	<takeOff, {identifier}, {response}>	---
---	<moveDown, {identifier, distance}, {response}>	---
---	<moveUp, {identifier, distance}, {response}>	---
<exit, {token}, {result}>	<quit, {identifier}, {result}>	---
<getWeather, {zipCode}, {weatherInfo}>	---	<getTemperature, {zipCode}, {temperature}> <getHumidity, {zipCode}, {humidity}> <getWind, {zipCode}, {windSpeed, windDirection}>
<getForecast, {zipCode, operatorName, period}, {weatherInfo}>	---	<getForecast, {zipCode, period}, {weatherData}>
<exit, {operatorName}, {result}>	---	

Figure 7: Ontological Correspondences

of executions of cycles making the traces finite.
The approach consist of three phases or steps: abstraction, matching, and synthesis. The *Abstraction* takes as input the NSs models and the subset of the domain ontology they refer to, and identifies the NSs common language through the ontologies. The common language makes NSs behavior comparable to reason on them. The *Matching* checks the NSs behavioral compatibility, i.e., that the two systems can synchronize at least on one trace reaching one of their respective final state. This step identifies possible mismatches to be reconciled while also taking into account a goal (if specified). Finally, the *Synthesis* produces a (intermediary) mediator that addresses the identified mismatches between the two NSs.

-DePer- This enabler perform dependability and performance analysis through stochastic model-based approach. DePer is composed by five main functional modules [2]: Builder, Analyser, Evaluator, Enhancer and Updater. The *Builder* module takes as input the specification of the Connected system. This specification is given as eLTS annotated with non-functional in-

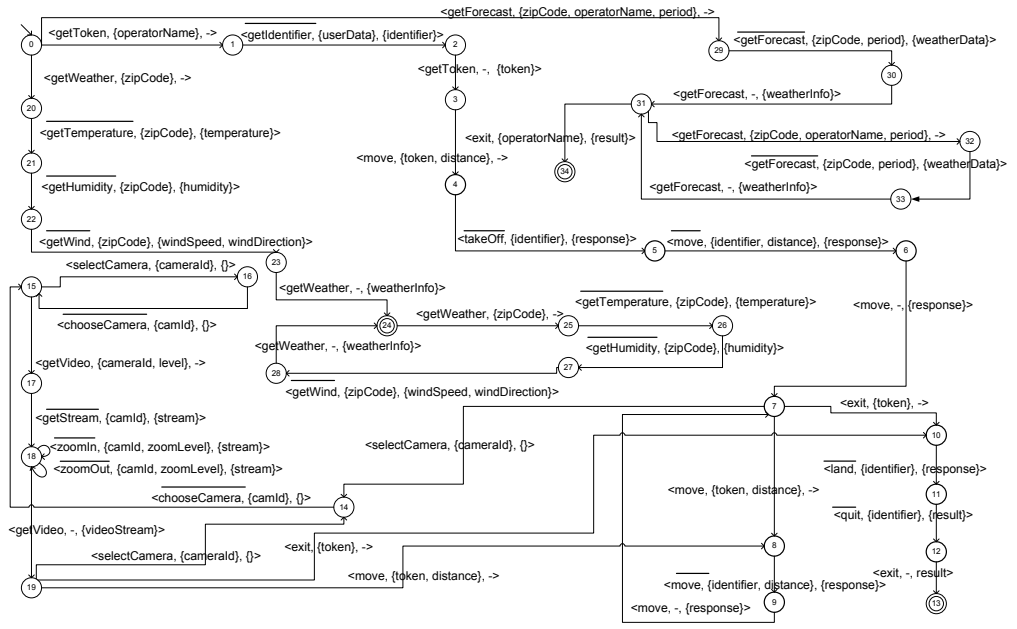


Figure 8: Behaviour of the Mediator

formation necessary to build the dependability and performance model of the CONNECTed system. The model is built up using the Stochastic Activity Networks (SAN) formalism [31]. The *Analyser* module extends the model developed by the Builder with reward functions suitable to quantitative assessment of the dependability and performance properties required by the NSs. For the assessment purpose, this module exploits Möbius [13]. The *Evaluator* module is in charge of checking whether the analysis results match with the properties required by the networking systems willing to communicate, or not. The *Enhancer* module is activated when the dependability and performance requirements are not satisfied, both at pre-deployment and at run-time, and try to enhance the CONNECTor with dependability mechanisms in order to satisfy the requirements. Finally, the *Updater* module, which is triggered only at run-time, continuously receives a flow of data for the parameters of interest related to the metrics under analysis, in order to refine the accuracy of model parameters through on-line observations and to perform new analysis on the updated model, by reactivating the cycle on the

Builder, Analyser and Evaluator modules.

4.2. Our Approach @ Pre-deployment Time

Our approach takes as input systems' specification including:

- the applications behavior (as enhanced Labeled Transition System, i.e., eLTS) including actions and input/output data
- the application domain ontology and sub-ontologies of it describing the meaning of the applications actions and data
- a goal (if not specified each trace of each NS is considered to be a goal)
- a bound on the number of executions of cycles in the applications behavior.

The first computation is done by the Synthesis module by reasoning on the input and following the three phases above described. Then, a mediator is automatically synthesized solving discrepancies and enabling the functional interoperation among the heterogeneous NSs.

Considering our case study, this step takes as input: the eLTSs of Command and Control Center (Figure 3), Weather Service (Figure 4), UAV with its integrated Video Camera (Figure 5); the GMES domain ontology and its sub-ontologies describing the applications actions and data to identify their common language (Figure 7). The output is the mediator of Figure 8.

After, DePer automatically builds the dependability and performance model of the Connected system, through the Builder module. It takes as input: the eLTS of the intermediate mediator synthesized during the previous stage, the annotated non functional data information of the NSs (i.e., the time to complete, the firing probability, and the failure probability of each labeled transition), and the dependability and performance requirements. DePer automatically translates the eLTS models of the networked systems and of the mediator into SAN models [32] by also considering the non functional data information during the translation, then automatically build the Connected system model. With respect to the forest fire scenario, we recall that the input eLTS are: the intermediate mediator of Figure 8, the Command and Control Center shown in Figure 3, the UAV illustrated in Figure 5, and the Weather Service shown in Figure 4. Further, we recall that the non functional requirements are: *latency* that represents a performance indicator

and is measured from when the Command and Control Center sends one of the possible order to move ($<\overline{move}, token, distance, ->$) to when it receives an acknowledgment ($<\overline{move}, -, response >$); *coverage* that represents a dependability indicator and is given by the percentage of weather data the Command and Control Center correctly receives from the Weather Service.

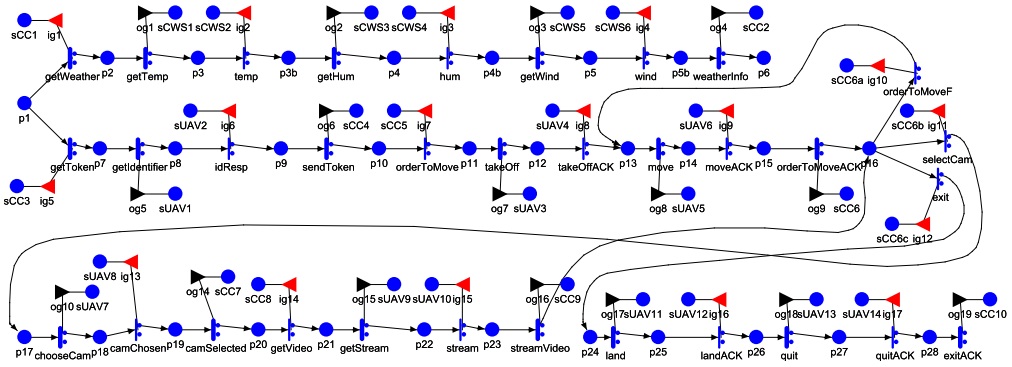


Figure 9: SAN model of the CONNECTor

The subsequent phase consists in performing the stochastic model-based analysis, by means of the Analyser module, in order to assess the desired non functional properties. The analysis results are then verified, by the Evaluator module, to check whether them match with the requirement of the networking systems willing to communicate, or not. At this point, two different situations may occur: (*i*) the dependability and performance requirements are satisfied; (*ii*) the analysis results do not match with the requirements. In both cases DEPER provides to the Synthesis module a feedback about the outcome of the analyses and, in case of unsatisfied dependability and performance requirements, the Enhancer module, which is in charge to select one of the existing mechanisms, proposes how to enhance the CONNECTor by including dependability mechanisms as countermeasures useful to contrast failure modes affecting dependability and/or performance metrics.
Considering our case study, the DEPER analysis returns as output a positive feedback with respect to the performance requirement, i.e., the CONNECTed system satisfies it; while it returns a negative feedback about the dependability requirement that is not satisfied. Hence, an enhancement is needed at this stage to properly enrich the mediator to let the CONNECTed system satisfy also the dependability requirement (if possible).
The output of this collaborative computation between Synthesis and DEPER

at pre-deployment time, given the initial systems knowledge, is an enhanced CONNECTor satisfying functional, performance, and dependability requirements. Regarding the case study, the output of this stage is the enhanced CONNECTor including the dependability mechanism as shown in Figure 10.

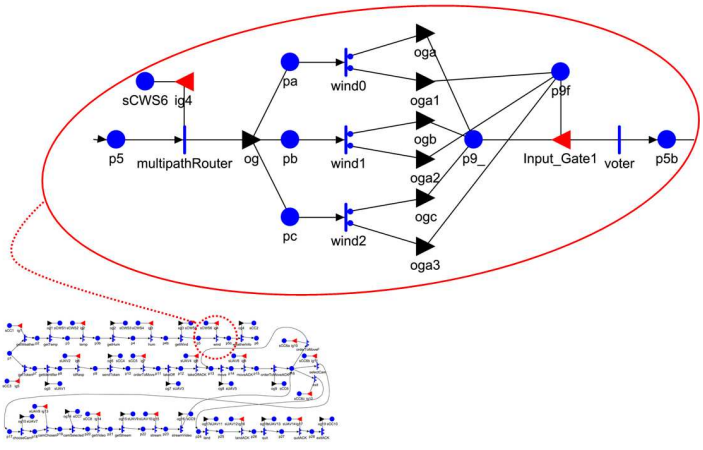


Figure 10: SAN CONNECTor model highlighting the portion where to integrate the Majority Voting mechanism (represented inside the big ellipse)

Analysis on the Forest Fire Case Study. Based on the above described scenario, we performed through Möbius [13] the analyses on the CONNECTed system in order to check if the synthesized CONNECTor satisfies the dependability and performance requirements. As mentioned above, the measures assessed in the evaluation are coverage and latency.

The first analysis, is related to dependability concerns and is intended to assess the measure of coverage, i.e., the percentage of weather data that the Command and Control Center correctly receives from the Weather Service. The results obtained from the pre-deployment analysis are shown in Figure 11, where the trend of the coverage is plotted (on the y axis) at increasing values of failure probability of the communication channel (on the x axis). Moreover, the coverage threshold is shown in the figure at the value of 0.90 (i.e., the 90% of all the received data is correctly received), as specified in the C2-A requirement. By observing the analysis results reported in the figure, it is possible to note that the (intermediate) synthesized CONNECTor does not satisfy the dependability requirement for any value of failure probability.

As previously said, an enhancement is performed on such (intermediate) CONNECTor by properly including a *Majority Voting mechanism* [22] (briefly described in Section 2) and new analysis are performed. The analysis results obtained on the *enhanced model* (including the original NSs and the enhanced CONNECTor), are also shown in Figure 11. It is possible to note that the use of this dependability mechanism allows to satisfy the coverage requirement when the failure probability is less than 0.2.

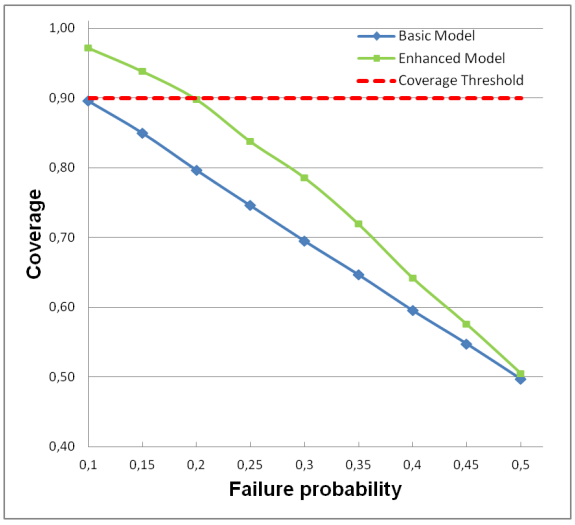


Figure 11: Coverage assessment as a function of failure probability of the communication channel

4.3. Our Approach @ Run-time

At *run-time* the applications and the previously synthesized mediator, also equipped with probes to monitor the CONNECTed system [3], are deployed and running on some devices. The probes, deployed into the CONNECTor, monitor the messages exchanged among the NSs involved into the communication and related to the metrics of interest.
When a performance or dependability violation is identified by the Updater module, through the probes, by reasoning on the new systems specification (that is the input systems specification modified through their run-time observation), DEPER selects the proper dependability mechanism that can be employed among those available, according to some internal predefined policies. The Builder module updates the model by including the selected

enhancing mechanism and new analyses are triggered by the Analyser module to verify whether the enhanced CONNECTor fulfills the dependability and performance requirements. When the employed mechanism is able to contrast the problem/violation, DEPER triggers the Synthesis by providing it the needed information to properly *enrich* the previously synthesized CONNECTor with the suggested mechanism. The output of this collaborative computation at run-time, given some performance or dependability violation notification, is a new CONNECTor satisfying the functional, performance, and dependability requirements.

Analysis on the Forest Fire Case Study. Considering our case study, the DEPER analysis returns feedback about the latency requirement is that it is not satisfied, and hence an enhancement is needed also at this stage. The output of this collaborative computation at run-time, given some initial systems knowledge, is an enhanced CONNECTor satisfying functional, performance, and dependability requirements.

In particular, according to the given definition of the latency, and based on the characteristics of the scenario, we have to point out that the first order of movement from C2-A is carried out from the UAV after the takeOff operations, so that the latency in this first phase is certainly greater than the flight phase.
We have also to note that the measure of latency, in this case study, is strictly related to the communication efficiency of the channels between the CONNECTor and the networked systems.

Once the Updater module updated the model with real value of the CONNECTed system, a second analysis was needed at run-time. Figure 12 shows the results obtained through the analyses on latency at varying of the rate of the distribution between 0.1 and 1 (on the x axis), which represents the transmission rate of the communication channel.

The analysis results show that the requirement, set to the value 5 *time units*, is never satisfied during the take off phase, while during the flight phase it is not meet for values of distribution rate less than 0.5. Some mechanisms is needed to enhance to connected system. The candidate mechanisms to improve the CONNECTed system with respect to the strong timing constraints, is the *Probing* (briefly described in Section 2). In Figure 12 are also shown the analysis performed on the model enhanced with the probing mechanism. It can be noticed that for values of distribution rate greater than 0.8, the CONNECTed system is be able to satisfy the requirement also during the take off phase. Figure 13 shows the SAN CONNECTor model where are highlighted

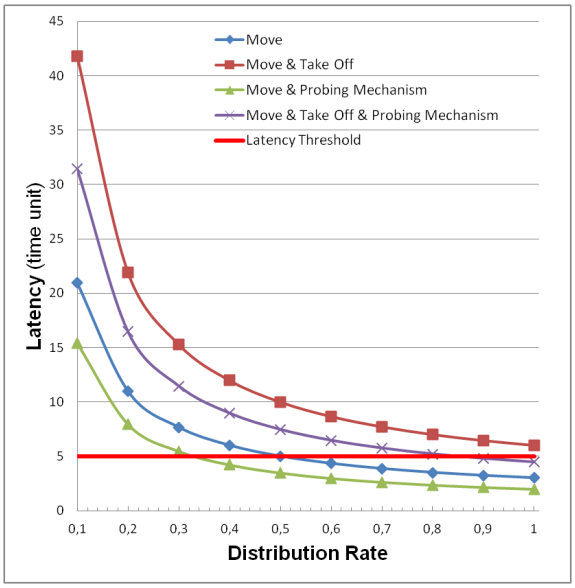


Figure 12: Latency assessment at varying of the rate of the distribution

the portion where to integrate the Probing mechanism and the mechanism itself (inside the big ellipse).

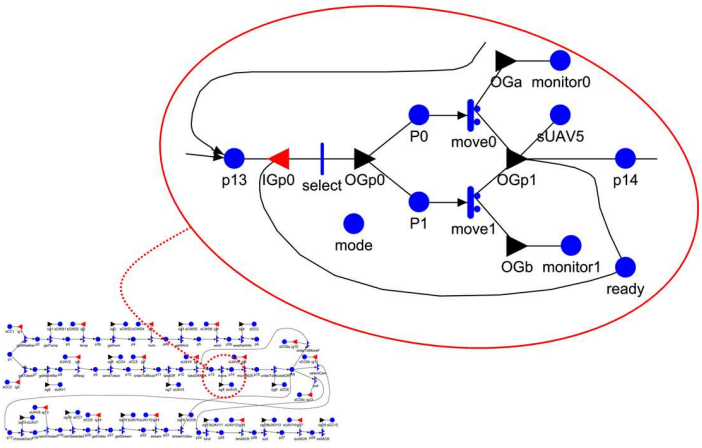


Figure 13: SAN CONNECTor model highlighting the portion where to integrate the Probing mechanism (represented inside the big ellipse)

5. Related Work

A big effort has been devoted in the literature to the investigation of the interoperability problem in the form of supervisory control synthesis [7], discrete controller synthesis [28], component adaptors [6], protocol conversion [8], [20], [25], converter synthesis [27] to mention few.

The theory of mediator, on which we build upon, is closely related to the seminal paper by Yellin and Strom on protocol adaptor synthesis [36]. They propose an adaptor theory to characterize and solve the interoperability problem of augmented interfaces of applications. Yellin and Strom formally define the checks of applications compatibility and the concept of adapters. Furthermore, they provide a theory for the automated generation of adapters based on interface mapping rules, which is related to our common language of protocols found through the domain ontology.

In more recent years an increasing attention has been payed in the Web Service area where many works are related to our synthesis of mediators for some aspect.Among them, papers [18, 19] propose a formal model to describe services and adapters and to automatically generate adapters. The work [15] presents an approach to specify and synthesize adapters based on domain-specific transformation rules and by using existing controller synthesis algorithms implemented in the Marlene tool[4]. The paper [21] on behavioral adaptation proposes a matching approach based on heuristic algorithms to match services for the adapter generation taking into account both the interfaces and the behavioral descriptions. Moreover, the Web services community has been also investigating how to actually support service substitution to enable interoperability with different implementations of a service (e.g., due to evolution or provision by different vendors). Our mediator synthesis work relates, for instance, to [10] by sharing the exploitation of ontology to reason about interface mapping and the synthesis according to such mapping. Their approach can be seen as an instance of ours.

Concerning combined approaches taking into account both functional and non functional issues, we can mention papers [26], [34], and [35]. This latter proposes an approach to automatically derive adaptors in order to assemble correct by construction real-time systems from COTS. The approach takes into account interaction protocols, timing information, and QoS constraints to prevent deadlocks and unbounded buffers. The synthesized adaptor is

[4] `http://service-technology.org/tools/marlene`

then a component that mediates the interaction between the components it supervises, in order to harmonize their communication. The purpose of our approach is similar to that in [35] since we both aim at synthesizing a mediator reconciling protocols, but our setting is quite different with respect to theirs. Indeed, our focus is mainly on solving protocols discrepancies to allow protocols synchronization to satisfy performance and dependability requirements also, while they focus more on timing and deadlock issues while composing COTS real-time components.

Spitznagel and Garland in their work [34] present an approach to formally specify connector wrappers as protocol transformations, modularizing them, and reasoning about their properties, with the aim to resolve component mismatches. In their vision a wrapper is new code interposed between component interfaces and communication mechanisms and its intended effect is to moderate the behavior of the component in a way that is transparent to the component or the interaction mechanism. Instead, a connector wrapper is a wrapper that address issues related to communication and compatibility including things such as changing the way data is represented during communication, the protocols of interaction, the number of parties that participate in the interaction, and the kind of communication support that is offered for things like monitoring, error handling, security, and so on. Their approach is to formally specify connector wrappers by means of a process algebra as a set of parallel process (one for each connector's interface and one for the glue) and to produce new connectors converting the protocol defining the first connector wrapper into a new protocol defining an altered connector by adding and modifying processes. Protocol transformations may include redirecting, recording and replaying, inserting, replacing, and discarding particular events yielding benefits like composability and reusability.

Moreover Spitznagel in her Ph.D. thesis [33] illustrates a set of patterns of basic connector's transformations, i.e., enhancements. She also shows how these patterns can be compositionally applied to simple connectors in order to produce a number of more complex connectors. In particular she shows how to select and to apply such transformations in the domain of dependability and proposes a prototypal tool to semi-automatically derive new connectors as enhancements.

Regarding the synthesis of CONNECTors meeting both functional and performance concerns, some authors of this paper already conducted investigations and presented an approach in [14]. Such approach is complementary

with respect to the work presented in this paper because tackles different aspects. The work in [14] takes place at pre-deployment time, when heterogeneous NSs requesting interoperability with some performance requirement, trigger the need of a CONNECTor. Its aim is to produce a CONNECTor satisfying both the functional and non functional required characteristics. The approach produces an intermediate CONNECTor and, by considering specific strategies, suggests how to properly prune and/or deploy it in order to improve the connected system performances to target performance requirements.
The work of this paper, instead, takes place both at pre-deployment time and run-time. Its aim is to cope with functional problems together with performance and dependability problems arising from the execution environment due to the uncertainties about the knowledge of the environment itself at pre-deployment time and the evolution of the working context. The approach at pre-deployment time, by considering specific mechanisms, suggests how to properly enrich an intermediate CONNECTor in order to improve the connected system performances and dependability -if needed. When a violation notification is identified at run time, the approach identify the proper mechanism to solve the problem and trigger the synthesis that take proper actions.

Stochastic model-based approaches for quantitative analysis of performance and dependability aspects have been largely developed along the last decades and documented in a large literature review on this relevant issue. A survey of the most popular ones can be find in [5], [24]. Commonly, the choice of the most appropriate model, to be used for this purpose, depends on several factors including the complexity of the system to be analyzed, the measures, the attributes and the measures to be evaluated, the accuracy required, and the resources available for the study. In this work the performance and dependability model of the CONNECTed system is specified with Stochastic Activity Networks (SANs), a generalization of Stochastic Petri Nets introduced and define in [31], [23].

In the last decade, a large number of studies addressed the problem of automated dependability analysis through the transformation of models. Automatic/automated methods from system specification languages to modelling languages amenable to perform dependability analysis has been recognised as an important support for improving the quality of systems. In [4] the authors present a development of an integrated environment to support the early phases of system design, where design tools based on the UML (Uni-

fied Modeling Language) are augmented with transformation-based validation and analysis techniques.

A Modeling framework allowing the generation of dependability-oriented analytical models from AADL (Architecture Analysis and Design Language) models is described in [29].

Several tools have been developed to support the definition of model-based transformations. The Viatra tool [12] automatically checks consistency, completeness, and dependability requirements of systems designed using the Unified Modeling Language. The Genet tool [9] allows the derivation of a general Petri net from a state-based representation of a system. The ADAPT Tool supports model transformations from AADL Architectural Models to Stochastic Petri Nets [30]. However, in terms of enhancing the model-transformation environment with template models of basic fault tolerance mechanisms to allow automated assessment of enhanced, fault tolerant designs, it seems a rather new research direction.

6. Conclusions

Diversity characterizing heterogeneous systems that are dynamically available in the networked environment is a richness. To be able to gain from it, requires to be able to cope with interoperability problems without a-priori knowledge of the systems, and with a degree of flexibility. We already proposed as solution an approach to the automated synthesis of CONNECTors (or mediators) between heterogeneous Networked Systems for their *functional interoperability* at application layer.

In this paper we presented: an approach to *enhance* the CONNECT*ors* taking into account performance and dependability aspects, a CONNECT*or adaptation process*, related to the performance and dependability mechanisms, spanning pre-deployment time and run-time, and a stochastic model-based *implementation* of the performance and dependability analysis.

Future investigations we plan to do concern: the study of techniques to apply on the system at run-time when a violation is detected; a more extensive validation of the approach on a number of case studies to precisely outline the typology of systems and problems we are able to mange automatically; a complete implementation of the overall approach.

Acknowledgements

The work is partly supported by the Connect European project No 231167.

References

[1] Baader F, Calvanese D, McGuinness DL, Nardi D, Patel-Schneider PF (2003) The Description Logic Handbook. Cambridge University Press

[2] Bertolino A, Calabrò A, Di Giandomenico F, Nostro N (2011) Dependability and performance assessment of dynamic connected systems. In: Bernardo M, Issarny V (eds) Formal Methods for Eternal Networked Software Systems, LNCS, vol 6659, Springer, pp 350 – 392

[3] Bertolino A, Calabro' A, Di Giandomenico F, Nostro N, Inverardi P, Spalazzese R (2012) On-the-fly dependable mediation between heterogeneous networked systems. In: ICSOFT 2011, CCIS 303, Springer-Verlag Berlin Heidelberg, 2012, pp 20–37

[4] Bondavalli A, Cin MD, Latella D, Majzik I, Pataricza A, Savoia G (2001) Dependability Analysis in the Early Phases of UML Based System Design. Journal of Computer Systems Science and Engineering 16(5):265–275

[5] Bondavalli A, Chiaradonna S, Giandomenico FD (2005) Model-based evaluation as a support to the design of dependable systems. In: Diab HB, Zomaya AY (eds) Dependable Computing Systems: Paradigms, Performance Issues, and Applications, Wiley, pp 57–86

[6] Bracciali A, Brogi A, Canal C (2005) A formal approach to component adaptation. J Syst Softw 74

[7] Brandin B, Wonham W (1994) Supervisory control of timed discrete-event systems. IEEE Transactions on Automatic Control 39(2)

[8] Calvert KL, Lam SS (1990) Formal methods for protocol conversion. IEEE Journal on Selected Areas in Communications 8(1):127–142

[9] Carmona J, Cortadella J, Kishinevsky M (2009) Genet: A tool for the synthesis and mining of petri nets. In: ACSD '09, IEEE Computer Society, Washington, DC, USA, pp 181–185, DOI http://dx.doi.org/10.1109/ACSD.2009.6

[10] Cavallaro L, Di Nitto E, Pradella M (2009) An automatic approach to enable replacement of conversational services. In: ICSOC/ServiceWave

[11] CONNECT Consortium (2012) Deliverable 6.3– Experiment Scenarios

[12] Csertan G, Huszerl G, Majzik I, Pap Z, Pataricza A, Varro D (2002) Viatra - visual automated transformations for formal verification and validation of uml models. In: 17th IEEE International Conference on Automated Software Engineering (ASE'02), pp 267–270

[13] Daly D, Deavours DD, Doyle JM, Webster PG, Sanders WH (2000) Möbius: An extensible tool for performance and dependability modeling. In: Haverkort BR, Bohnenkamp HC, Smith CU (eds) 11th Int. Conf., TOOLS 2000, LNCS, vol 1786, Springer Verlag, pp 332–336

[14] Di Marco A, Inverardi P, Spalazzese R (2012) Synthesizing connectors meeting functional and performance concerns. Tech. rep.

[15] Gierds C, Mooij AJ, Wolf K (2012) Reducing adapter synthesis to controller synthesis. IEEE Trans Services Computing 5(1):72–85

[16] Inverardi P, Issarny V, Spalazzese R (2010) A theory of mediators for eternal connectors. In: Proceedings of ISoLA 2010, Springer

[17] Inverardi P, Spalazzese R, Tivoli M (2011) Application-Layer Connector Synthesis. In: Formal Methods for Eternal Networked Software Systems (SFM'11), vol 6659, pp 148–190

[18] Jiang J, Zhang S, Gong P, Hong Z (2011) Message dependency-based adaptation of services. In: APSCC, pp 442–449

[19] Jiang J, Zhang S, Gong P, Hong Z (2011) Service adaptation at message level. In: SERVICES, pp 87–88

[20] Lam SS (1988) Correction to "protocol conversion". IEEE Trans Software Eng 14(9):1376

[21] M Nezhad HR, Xu GY, Benatallah B (2010) Protocol-aware matching of web service interfaces for adapter development. In: Proceedings of the 19th international conference on World wide web, ACM, New York, NY, USA, WWW '10, pp 731–740, DOI http://doi.acm.org/10.1145/1772690.1772765, URL `http://doi.acm.org/10.1145/1772690.1772765`

[22] Masci P, Nostro N, Di Giandomenico F (Geneva, Switzerland, September 2011) On enabling dependability assurance in heterogeneous networks through automated model-based analysis. In: Proc. Third International Workshop, SERENE 2011, Sringer, LNCS, vol 6968, pp 78 – 92

[23] Movaghar A, Meyer JF (1984) Performability modelling with stochastic activity networks. In: 1984 Real-Time Systems Symposium, IEEE Computer Society Press, Austin, TX, pp 215–224

[24] Nicol DM, Sanders WH, Trivedi KS (2004) Model-based evaluation: from dependability to security. IEEE Transactions on Dependable and Secure Computing 1:48–65

[25] Okumura K (1986) A formal protocol conversion method. In: SIGCOMM, pp 30–37

[26] Oster ZJ, Santhanam GR, Basu S (2011) Identifying optimal composite services by decomposing the service composition problem. In: ICWS, pp 267–274

[27] Passerone R, de Alfaro L, Henzinger TA, Sangiovanni-Vincentelli AL (2002) Convertibility verification and converter synthesis: two faces of the same coin. ICCAD '02, pp 132–139

[28] Ramadge P, Wonham W (1987) Supervisory control of a class of discrete event processes. Siam J Control and Optimization 25(1)

[29] Rugina AE, Kanoun K, Kaniche M (2007) A system dependability modeling framework using aadl and gspns. In: Lemos R, Gacek C, Romanovsky A (eds) Architecting Dependable Systems IV, Lecture Notes in Computer Science, vol 4615, Springer Berlin Heidelberg, pp 14–38, DOI 10.1007/978-3-540-74035-3_2, URL `http://dx.doi.org/10.1007/978-3-540-74035-3_2`

[30] Rugina AE, Kanoun K, Kaniche M (2008) The adapt tool: From aadl architectural models to stochastic petri nets through model transformation. In: Seventh European Dependable Computing Conference, EDCC-7 2008, Kaunas, Lithuania, 7-9 May 2008, IEEE Computer Society, pp 85–90, DOI http://dx.doi.org/10.1109/EDCC-7.2008.14

[31] Sanders WH, Meyer JF (2002) Stochastic activity networks: formal definitions and concepts pp 315–343

[32] Sanders WH, Meyer JF (2002) Stochastic Activity Networks: formal definitions and concepts pp 315–343

[33] Spitznagel B (2004) Compositional transformation of software connectors. PhD thesis, Carnegie Mellon University

[34] Spitznagel B, Garlan D (2003) A compositional formalization of connector wrappers. In: ICSE, pp 374–384

[35] Tivoli M, Fradet P, Girault A, Gößler G (2007) Adaptor synthesis for real-time components. In: TACAS, pp 185–200

[36] Yellin DM, Strom RE (1997) Protocol specifications and component adaptors. ACM Trans Program Lang Syst 19

Networks of heterogeneous and dynamic interoperable systems: an approach to enhance dependability and performance properties

Felicita Di Giandomenico[1], Massimiliano L. Itria[1],
Paolo Masci[2], Nicola Nostro[1]

[1] ISTI-CNR, Pisa, Italy
{felicita.digiandomenico,massimiliano.leone.itria,nicola.nostro}@isti.cnr.it

[2] Queen Mary University of London, UK
paolo.masci@eecs.qmul.ac.uk

Abstract. Approaches to dependability and performance are challenged when systems are made up of networks of heterogeneous applications/devices, especially when operating in unpredictable open-world settings. The research community is tackling this problem and exploring means for enabling interoperability at the application level. The EU project CONNECT has developed a generic interoperability mechanism which relies on the on-the-fly synthesis of "CONNECTors", that is software bridges that enable and adapt communication among heterogeneous devices. Dependability and Performance are relevant aspects and a model-based analysis framework has been addressed for enabling the synthesis of dependable CONNECTors. In a previous work, we have identified mechanisms that can be used as the basis for the definition of a library of generic patterns (*dependability mechanisms*) suitable for enhancing the dependability of CONNECTors when required by the application. Here, we extend it by identifying *generic strategies* that automate the selection of an appropriate dependability mechanism and of those elements in the software bridge to which the mechanism brings higher benefits. A case study based on a global monitoring system for environment and security (GMES) is also included to show the application of our solution.

1 Introduction and motivation

The classic and well understood way of building dependable systems [6] is based on the application of rigorous development methods. Special programming techniques are used for software, such as model-driven development [11], and specific architectures are used for hardware, such as modular redundancy [4]. Dependability-critical domains, such as avionics and power plants, *require by law* the adoption of these techniques, and define standards that must be followed.

This classic approach to dependability is challenged when critical systems are made up of networks of heterogeneous devices from different manufacturers. The

This paper is under refinement for submission to an international, peer-reviewed journal – December 10, 2012

GMES (Global Monitoring for Environment and Security) European Programme for the establishment of a European capacity for Earth Observation provides an excellent example of heterogeneity in interoperable applications and devices for critical applications. It started in 1998, and includes six main thematic areas: land monitoring, marine environment monitoring, atmosphere monitoring, emergency management, security and climate change. The emergency management service directs efforts towards a wide range of emergency situations; in particular it covers different catastrophic circumstances: floods, forest fires, landslides, earthquakes and volcanic eruptions and humanitarian crises. As another example, in the healthcare domain, currently there is not a standard for medical device interoperability. Nevertheless, this has not prevented the adoption in hospitals of networks of heterogeneous technologies. In some cases lack of interoperation just causes minor disturbances, e.g., patients not recognised by palm-sized wireless medical devices because they are not able to request on-the-fly this information to remote servers [19]. In other cases, problems are more serious, e.g., surgical fires caused by lack of dependable interoperation between electrosurgical devices and oxygen-delivery devices [10].

The problem is that standardised interaction at the application level is essentially *non-existent*. In fact, standards like Universal Serial Bus (USB) and IEEE 802.11 (WiFi) enable interoperability at a level *lower* than the application logic. The consequence of this is that heterogeneous networked devices might be able to interoperate but at the same time they might not be enabled to fully benefit from each other's services. This situation might create serious problems; e.g., in safety-critical systems safety interlocks defined at the application level may be ignored or overridden. Even more challenging is the situation where the heterogeneous systems willing to interoperate have a dynamic and evolving behaviour, thus requiring adaptation and evolution of the interoperation means itself.

The research community is exploring means for enabling interoperability at the application level. Ad hoc solutions for specific applications are popular research topics, e.g., "smart alarms" [15, 28], and "medical device dongle" [3]. Generic solutions that cross-cut different application domains, on the other hand, are almost unexplored.

CONNECT (`http://www.connect-forever.eu`) is a research project that has been active in exploring generic solutions to interoperability at the application level. The generic interoperability mechanism studied in the project relies on the dynamic synthesis of "CONNECTors", software bridges that enable and adapt communication among heterogeneous devices. In [27] it has been shown that a model-driven approach can be used to automatically generate such CONNECTors. That is, software for enabling interoperability at the application level can be automatically generated from models describing the behaviour of applications executed on heterogeneous devices. A methodology has been also developed to adapt and enhance the CONNECTor i response to changes and evolution of the networked systems and their operation environment.

In our previous works [18, 20] we have developed a model-based analysis framework for enabling the synthesis of dependable CONNECTors. Recently,

in [21], we have identified mechanisms that can be used as the basis for the definition of a library of generic patterns (*dependability mechanisms*) suitable for enhancing the dependability of CONNECTors when required by the application. The need for enhancing a CONNECTor may arise at design time, based on the results of a stochastic model-based analysis that reveals whether non-functional requirements are satisfied or not, but also during the CONNECTor lifetime, due to the dynamic behaviour and evolution of the networked systems requesting to be CONNECTed. In this work, we complement the former work by identifying *generic strategies* that allow us to automate the selection of an appropriate dependability mechanism.

The contributions of this work are: (i) the definition of a generic method for selecting a dependability mechanism suitable to enhance the software bridge; (ii) the definition of a generic method for identifying elements in the software bridge that shall be replaced; (iii) a case study based on a global monitoring system for environment and security (GMES).

The presentation proceeds as follows. In Section 2, we provide an overview of the CONNECT framework, as this work is based on it. In Section 3, we illustrate the proposed generic methodology for selecting dependability mechanisms in networks of heterogeneous interoperable devices. In Section 4, we demonstrate the benefits of the proposed approach within a case study based on a global monitoring system. The selected scenario is one of the demonstrative examples developed in the CONNECT project. Section 6 describes related work and draws the conclusions.

2 Background on the Connect project

The CONNECT approach relies on a framework where several active units operate in the network, among which: Discovery/Learning, Synthesis, Dependability, and Monitoring.

Discovery/Learning This unit gathers information about functionalities requested and provided by networked systems. Specifically, the unit discovers mutually interested devices, and retrieves information about their interface behaviours. The unit assumes that devices are *discovery enabled*, i.e., they provide a minimal description of their intent and functionalities. When a networked system just provides a partial specification of its behaviour, the Discovery/Learning unit completes the specification through a learning procedure (e.g., via model-based testing).

Synthesis This unit performs the dynamic synthesis of mediating CONNECTors to enable interoperation among devices willing to interact. The unit uses behavioural models built by Discovery/Learning (i) to identify mismatches between communication protocols employed by networked systems, and (ii) to generate software bridges that resolve the identified mismatches.

Dependability This unit supports the Synthesis unit in the definition of a software bridge that allows networked systems to interact in a dependable way.

Specifically, the unit performs a stochastic model-based analysis that predicts whether the overall CONNECTed system will meet given dependability requirements. If the analysis results reveal that the dependability requirements may not be satisfied by the software bridge generated by Synthesis, then the Dependability unit instructs Synthesis about enhancements that can be applied to the software bridge.

Monitoring This unit becomes operational when the software bridge is deployed. The unit continuously monitors the CONNECTor in order to update the other units of the CONNECT framework with run-time data. This allows adaptation and evolution of the software bridge.

The life-cycle of a CONNECTor starts with a networked device broadcasting a CONNECT *request*. This happens whenever the device requires a service. The CONNECT request contains a description of the functionalities the service should provide, together with a specification of dependability requirements. The request is processed as follows within the CONNECT framework:

1. Discovery/Learning captures the CONNECT request and looks for networked devices that can provide the requested service. If such a device is found, Discovery/Learning activates Synthesis to generate a suitable CONNECTor that enables interoperation (a *null* CONNECTor will be generated if the communication protocols of the two networked devices are already compatible).
2. Synthesis generates the specification of a mediating CONNECTor. This is done on the basis of the specification of the communication protocols. Before deploying the synthesised CONNECTor, Synthesis activates the Dependability unit to assess whether the CONNECTed system meets given dependability requirements.
3. Dependability performs a model-based evaluation of the CONNECTed system. When given dependability requirements are not met, the unit informs Synthesis about how the synthesised CONNECTor can be improved.
4. Once the CONNECTor is deployed, its runtime behaviour is monitored through the Monitoring unit, which will notify other units about changes.

3 Automated selection of a dependability mechanism

Three main aspects have significant relevance when selecting dependability mechanisms suitable to enhance CONNECTors: (i) constraints imposed by the application domain, e.g., applications providing remote-control services may have different constraints in terms of timing and tolerance to degraded quality from video-based applications; (ii) assumptions about fault models of devices during their operational life, e.g. transient faults or permanent faults; (iii) dependability metrics relevant to given dependability requirements, e.g., message delivery time, coverage of receivers.

Application constraints Application constraints may change (i) the system tolerance to degraded service, and (ii) the ability to deploy specific solutions.

In some cases this may hinder the benefits of certain dependability mechanisms. For instance, consider a dependability mechanism based on error-correction that can be used to detect transmission errors and reconstruct the original error-free data. This mechanism assumes that the original error-free data are available at the transmitter. Therefore, the mechanism can be used to enhance the dependability level of a CONNECTor only when such an hypothesis holds, i.e., either when the CONNECTor can be deployed on the transmitter side, or when a reliable channel exists between the transmitter and the infrastructural element with the deployed CONNECTor.

Failure assumptions Dependability and performance models are typically based on assumptions about the malfunctions, experienced by the system under analysis during its operational life, which may undermine the (dependability or performance) indicators under assessment. Of course, the nature of the assumed faults and/or failures guides the selection of countermeasures: those which are particularly effective to contrast them. A taxonomy of the faults and failures, from several facets, has been proposed in the dependability community [6]. Restricting to the five dependability and performance mechanisms we have described in [21], the interest is mainly in: (i) the persistence of the fault, that is whether the fault is transient (maybe, in bursts, but lasting a limited amount of time), or permanent; (ii) the objective of the fault, that is if it is accidental or intentionally introduced in the system; (iii) the failure domain, that is content failures when the content of the information delivered at the service interface deviates from implementing the system function, or timing failures when the time of arrival or the duration of the information delivered at the service interface deviates from implementing the system function. We will discuss further below how the dependability mechanisms we adopt relate with these faults and failures assumptions.

Dependability metric Dependability metrics guide the definition of the assessment model, which has to faithfully include all the system aspects with a relevant impact on the measure under evaluation while abstracting or even neglecting all the other behaviors/phenomena with negligible impact, in order to keep the model as much as possible manageable and controllable. The measure also influences the choice of the dependability mechanism, characterised by differing structure and operational behaviors. In the context of CONNECT, we mainly addressed two categories of measure for stochastic quantitative analysis: performance-related one (including, e.g., variants of the latency indicator), and dependability-related one (including, e.g., different forms of coverage as percentage of networked services receiving/offering a service with respect to the full population).

In the following we illustrate a method to guide the selection of dependability mechanisms for enhancing interoperability among networked devices. The method is explained with respect to the CONNECT framework. In the literature, there is no evidence that relying on just one of the identified factors is the best

choice. Therefore, our proposed selection method will consider a combination of these identified factors.

3.1 Proposed method to select a dependability mechanism

The CONNECT project has identified that ontologies [12] are the pillar to establish a common understanding of the specification of networked systems. Therefore, the matchmaking of networked systems towards their connection is first determined according to the ontology-based semantic matching of their affordances. Given semantically matching affordances, their associated protocols must be checked for the potential to coordinate, possibly under some mediation implemented by the supporting CONNECTor. Advanced matching relations and supporting algorithms are being investigated within the project, so as to facilitate connection between heterogeneous systems. According to the classification of CONNECTors adopted by the project, and the related reference middleware ontology, the following reference coordination models have been identified: client-service, message-orientation, publish-subscribe and shared memory.

Following the same ontology-based approach, the general method we propose consists of the following steps.

1. Eliciting an ontology-based characterisation of the dependability mechanisms (at the moment, limited to the five mechanisms introduced so far, but open to include new ones in the future) of the properties of each mechanism with respect to the three aspects of application constraints, measure under analysis, and fault/failure assumptions. For each of them, the categories previously discussed are considered at the moment.
2. When a dependability mechanism is required, to overcome weaknesses in satisfying the dependability or performance requirements on the CONNECTed system as revealed by the dependability analysis, operate an ontology-based matching to derive the subset of mechanisms that have at least one correspondence in terms of the three characterising aspects.
3. If the resulting subset includes only one mechanism having the highest matches, then select that mechanism as the most appropriate to apply.
4. If more than one mechanism show the highest matches, maybe on different aspects (e.g., one mechanism matches on fault/failure assumptions and application constraints, while the other one matches on application constraints and measure), then a prioritisation among the three aspects would be necessary to rank the mechanisms in the subset.

The following Table 1 is an example of instantiation when taking into account slightly revisited versions of the dependability mechanisms described in [21]: retry, probing, majority voting, error correction, and parallel retry. For each dependability mechanism we identify the three major impacting aspects that must be taken into account when selecting a suitable mechanism for enhancing the CONNECTor. To better understand the content of Table 1, a summary of each included mechanism is provided in the following.

- **Retry.** This mechanism, widely adopted in communication protocols, consists in re-sending messages that get corrupted or lost during communications, e.g., due to transient failures of communication links. A typical implementation of the retry mechanism uses time-outs and acknowledgements: after transmitting a message, the sender waits for a message of the receiver that acknowledges successful communication. If the acknowledgement is not received within a certain time interval, the sender assumes that the communication was not successful, and re-transmits the message.
- **Probing.** This mechanism exploits redundant paths and periodic *keep-alive* messages for enabling reliable communication in face of path failures. The basic idea is to continuously collect statistics about the characteristics of the communication channels, and to select the best channel on the basis of such statistics.
- **Majority Voting.** This is a fault-tolerant mechanism that relies on a decentralised voting system for checking the consistency of data. Voters are software systems that constantly check each other's results, and has been widely used for developing resilient systems in the presence of faulty components. In a network, voting systems can be used to compare message replicas transmitted over different channels.
- **Error Correction.** This mechanism deals with the detection of errors and re-construction of the original, error-free data. A widely used approach for enabling hosts to automatically detect and correct errors in received messages is *forward error correction* (FEC). The mechanism requires the sender host to transmit a small amount of additional data along with the message.
- **Parallel Retry.** This is similar to the Retry mechanism already introduced, but exploits channels redundancy (instead of successive retries on the same channel) to speed up the communication time.

Note that a more precise identification of dependability mechanism is also possible when performing a sensitivity analysis. In fact, a sensitivity analysis would point out which model parameters mostly impact the dependability measure that must be improved (e.g, time to complete a transition and its failure probability). We will discuss this in the implementation section.

3.2 Where to apply the dependability mechanism

The most accurate method to identify element(s) of the CONNECTor that should be enhanced through the selected dependability mechanism is based on a systematic exploration of the benefits of the mechanism to each individual element. The element for which the benefit is the highest is the best candidate. While very accurate, this approach is time consuming. We have developed an alternative method that provides a good trade-off between time and accuracy in several practical cases. The idea of the approach is that an exploratory investigation based on simple probabilistic formulations can be performed with the aim to find out where to apply the dependability mechanism. The approach is tailored

	Characterising aspects		
Mechanism	**Application Constraints**	**Fault/Failure Assumption**	**Evaluated Measure**
Retry	soft timing constraints	transients, short duration/omission failures	dependability
Probing	generic constraints	transients and permanent/omission failures	dependability & performance
Majority Voting	generic constraints	transients and permanent/omission and value failures	dependability
Error Correction	medium/soft timing constraints	transients/value failures	dependability (less performance)
Parallel Retry	generic constraints	transients and permanent/omission failures	performance (less dependability)

Table 1. Suitability of the developed dependability mechanisms wrt the three identified impacting aspects

to two specific cases: dependability-related metrics and performance-related metrics.

Before detailing the approach, we briefly discuss our setting. We consider a model-based analysis where communication among heterogeneous devices is specified in terms of timed activities. Each timed activity a_i is characterised by a time to complete t_i and a failure probability q_i, independently from the others. For the illustrative purpose of this work, we assume that dependability-related metrics are mainly influenced by failure probability, while performance-related metrics are mainly influenced by the time to complete. Under these assumptions, two automated strategies (one for dependability-related metrics, one for performance-related metrics) can be defined for identifying elements in the CONNECTor that should be enhanced with dependability mechanisms.

Dependability-related metrics case. We start with the simple case where: i) the metric under analysis is a function of the failure probabilities of all the activities in the model representing the CONNECTor; ii) each activity can either be successfully completed or fails. This means that there is only one path for successful executions of the CONNECTor. Figure 1(a) illustrates this case.

Let's consider that N activities are included in the model of the CONNECTor under analysis. Then, the selection of the activity to be enhanced is simply determined through the following steps:

1. for each activity a_i, determine the new value $\bar{q}_i$ resulting from applying the dependability mechanism to activity a_i;

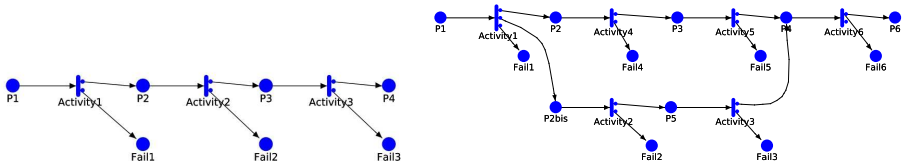


Fig. 1. Example of a basic CONNECTor model (a), and of a more complex CONNECTor with two branches (b)

2. for each activity a_i, re-determine the value of the failure probability Q_i of the whole CONNECTor when using the new value $\bar{q}_i$. The general formulation of the CONNECTor failure probability Q is given by the expression: $q_1 + \sum_{i=2}^{N} q_i \cdot \prod_{j=1}^{i-1} (1 - q_j)$;
3. select the activity a_j whose new probability value $\bar{q}_j$ for which Q_i is the minimum among the N values of Q_i.

Just for illustrative purpose of the approach, let's refer to the model in Figure 1(a) and let's consider the starting failure probabilities q_i as reported in the following Table, where two dependability mechanisms have been considered for the enhancement purpose (Majority Voting using three channels and Retry executed two times). According to the values determined, enhancement of activity a_1 is decided when applying Majority Voting mechanism, while activity a_4 is selected in case of the retry mechanism.

q_i	$\bar{q}_i$(voting)	$\bar{q}_i$(retry)	Q_i(voting)	Q_i(retry)
0.2	0.104	0.04	0.6610432	0.636832
0.03	0.002646	0.0009	0.688825552	0.6882808
0.35	0.28175	0.1225	0.6655828	0.591436
0.4	0.352	0.16	0.6731488	0.576304

Table 2. Illustrative example of the selection approach for dependability-related metrics

Generalisations of this simple case can be performed in order to take into account that: i) that not all the activities a_i are involved in the dependability metric under analysis, and ii) more than one path is possible within the dependability model to reach a_i from the start activity (e.g., as illustrated in Figure 1(b)).

With respect to point i), the generalisation is easily performed by determining the new values $\bar{q}_i$ only for those activities a_i involved in the formula expressing the metric and consequently restricting the calculation of Q_i only to the $\bar{q}_j$ computed. Then, the last step remain the same.

The second generalisation does not require any actual modification to the described basic method. What changes is just the formula expressing the CONNECTor failure probability Q, which has to take into account the different paths which may lead to failure. The formulation is straightforward and omitted here to save space.

An interesting observation concerns the formula expressing the improvement in reliability gained through the application of the schemes for which a dependability model has been generated (e.g., those in Table 1), necessary to derive the new values $\bar{q}_i$ associated to activities a_i. It could be stored together with the mechanism model as a parametric reliability expression, and be efficiently adapted with the current probability failure parameter q_i of the activity at hand when a specific CONNECTor needs to be enhanced. This allows to speed up the application of the selection procedure.

Finally, by executing a campaign of experiments starting with basic failure probabilities q_i lower then 0.01, we found that the approach described above could be reasonably approximated by directly choosing the activity with the highest failure probability value. Again, this promotes efficiency of the selection process and this is actually the method currently implemented in the DEPER prototype.

Performance-related metrics case. In the case of performance-related metrics, the choice of the model elements to be enhanced is expected to be made among those representing communications with longer time to complete. We recall that performance metrics we are dealing with are classical metrics, typically the degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage, as defined in [1]. Therefore, with reference to our CONNECT context, the dependability mechanisms should act to improve transmission time essentially in presence of possible congestion of the communication channels, responsible for longer times. Among the dependability mechanisms shown in Table 1, Parallel Retry and Probing are the only ones adequate to address performance improvements: by resorting to additional resources (transmission channels in our context), with assigned a transmission rate taken from a probabilistic distribution, these mechanisms allow to exploit the most performable ones. E.g., the Probing mechanisms, by continuously monitoring the efficiency of both channels available for transmission, at each sending uses the most efficient one.

The approach to select the activity to improve is similar to the dependability-related metric case, but specular under the time domain. The steps are:

1. identification of those activities that are involved in the performance metric (unless we are dealing with an end-to-end metric, a subset of activities is typically involved);
2. for each activity a_i in the involved set, re-determine the value of the transmission time parameter $\bar{t}_i$ resulting from applying the dependability mechanism;

3. since it is expected that the performance-related metric is based on the summation of the execution time of the activities under consideration, select the activity a_j for which the difference between the original time parameter value t_i and the new one $\bar{t}_j$, as calculated at step 2., is the largest.

Similarly to Table 2, the following Table 3 elaborates a simple example for the performance-related case, using the Probing and Parallel Retry (using three channels) as dependability mechanisms.

t_i	$\bar{t}_i$(probing)	$\bar{t}_i$(parallel retry)	t_i-$\bar{t}_i$(probing)	t_i-$\bar{t}_i$(parallel retry)
4.664	4.53	1.589	0.134	3.075
2.49	2.302	0.794	0.188	1.696
1.747	1.497	0.529	0.25	1.218
1.302	1.296	0.454	0.006	0.848

Table 3. Illustrative example of the selection approach for performance-related metrics

From the table, it can be observed that the Parallel Retry results in much better values than those of the Probing, at the cost of a higher redundancy. The choice of the tuning of the mechanisms parameters (such as the number of additional channels for the Parallel Retry) depends on the level of performance that need to be guaranteed for the specific application at hand.

Some general observations are now provided that are relevant to the whole Section. First, dependability and performance properties are more and more required to be considered in strict relationship in a variety of critical applications. Therefore resorting to separate evaluation models would result inadequate, since they fail to capture the consequences of failures on the degradation of the performance. Performability metrics have been proposed [2] as a unification of performance and dependability, that is system's ability to work in the presence of errors and failures. In contrast with pure performance, which refers to how efficiently a system delivers a specified service assuming it is delivered correctly, with performability effects of faults are considered and, at the same time, failures due to faults are not the only relevant phenomena to assess. It is planned to extend this work to deal also with performability metrics.

Another aspect that we would like to point out is that, after the application of our method to select a model element for enhancement through the dependability mechanism (either for dependability- and performance-related measures), and the consequent extension of the CONNECTed system model for a new analysis phase, it could happen that the results are not yet satisfactory with respect to given requirements. This means that, iteratively, the selection of an additional element needs to be carried on repeating the same steps as before, until the requirement is met or the whole model has been enhanced without success. It would be interesting to investigate methods to predict, with a satisfactory level of accuracy, the minimum set of elements that would be necessary for require-

ment satisfaction within the next analysis phase. This would greatly reduce the exploration time. This research direction is part of our future research agenda.

4 GMES Case Study

In this section, we present our demonstrative scenario, based on the GMES (Global Monitoring for Environment and Security)[3] European Programme for the establishment of a European capacity for Earth Observation, in order to show how the proposed method can be applied to select proper dependability mechanism and where to apply them.

GMES services address six main thematic areas: Land Monitoring, Marine Environment Monitoring, Atmosphere Monitoring, Emergency Management, Security, and Climate Change. The GMES emergency management service provides all actors involved in the management of natural disasters. In particular, it covers different catastrophic circumstances: floods, forest fires, landslides, earthquakes and volcanic eruptions and humanitarian crises.

4.1 Forest-fire emergency

In this work, we concentrate on the Forest fire emergency situation [8]. The scenario describes the management of forest-fire, close to a border village and a factory between two different countries, Country A and Country B. Forest monitoring and forest fire management in the country A are the responsibility of Country A Command and Control fire operations centre (C2-A).

The Forest-fire scenario addresses different phases. We focus on the *Reinforcement integration phase*, where Country B provides an unmanned aerial vehicle (UAV) to Country A. The UAV is with various video cameras that allow to get a better view of the fire front close to the village in order to be able to proceed to its evacuation in time. Country B grants access to its weather forecast service in order to continuously provide information about temperature, humidity and wind of the area interested by the fire.

During the crisis phase, Country A's Command and Control fire operations centre (C2-A) is in charge of the management of the forest fire crisis. This phase involves a number of sensors and human actors. In case the fire goes wider and there is a direct threat to the village and the factory, Country A asks support from Country B (*reinforcement phase*). The reinforcement phase starts when support resources provided by Country B are deployed and controlled by C2-A. Once deployed on the emergency area, resources from Country B are seen by C2-A as resources available and usable during the Fire-fighting phase. Resources provided by Country B have the functionalities similar to those of Country A's resources, (e.g., UAVs provide high quality images or weather information of the area interested by the fire) but use different protocols.

Networked Systems involved in this scenario are: C2-A, the Weather Service, and the UAV with integrated video camera. For the illustrative purpose of this

[3] http://www.gmes.info/

work we consider just C2-A and the UAV, as this is sufficient to demonstrate the approach. We included the UAV instead of the Weather Service due to the fact that it provides a more rich scenario, and therefore more interesting to study.

The behaviour of the considered Networked Systems is described in the following subsections.

Command and control Centre of Country A. The Command and Control Centre represents the first Networked System which needs to use specific resources. When C2-A wants to access the resources equipped with one (or more) video camera(s) and operating in the field, like an unmanned ground vehicles (UGV), it first needs to authenticate with the resource sending a *getToken* message, after that it receives back a message containing the *token* useful to access the resources. Once C2-A has a token, it can instruct the resource to move forward, backward, left and right, by sending the corresponding message, which is followed by an acknowledge message confirming the movement. At the same time C2-A can select the camera installed on the resource through the message *selectCamera* and than it can receive the video stream with a specified zoom level, by sending the message *getVideo*.

UAV and integrated Video Camera of the Country B. The user of the UAV first needs to authenticate with the service by sending a `getIdentifier` request. After that it receives back an identifier (`idResp`) and can trigger the flight phase through the `takeOff` message. After the take off, the user can order the UAV to move left, right, front, back, up or down, or to land. For each order of movement, the UAV replies with an acknowledge message. Moreover, at anytime during the flight phase, the user can get from the integrated video camera the video stream showing in real time the specified area of interest, as well as to perform zoom-in and zoom-out.

Connecting C2-A and the UAV. The application behaviour of the CONNECTed system is depicted by the sequence diagram in Figure 2. Given the heterogeneity between the Command and Control Center and the UAV involved in this GMES scenario, it is necessary to synthesise on-the-fly a CONNECTor to allow the communications and the exchange of information between the two Networked Systems. A CONNECTor is therefore synthesised and analysed to assess whether it meets the dependability and performance requirements.

4.2 Analysis

From the specification of the CONNECTed system, we generated a model suitable to assess dependability and performance related metrics. This task is accomplished in the CONNECT project by an automated process implemented in the Dependability and Performance unit mentioned in Section 2. The adopted model formalism is SAN (Stochastic Activity Network) and the analysis is carried on

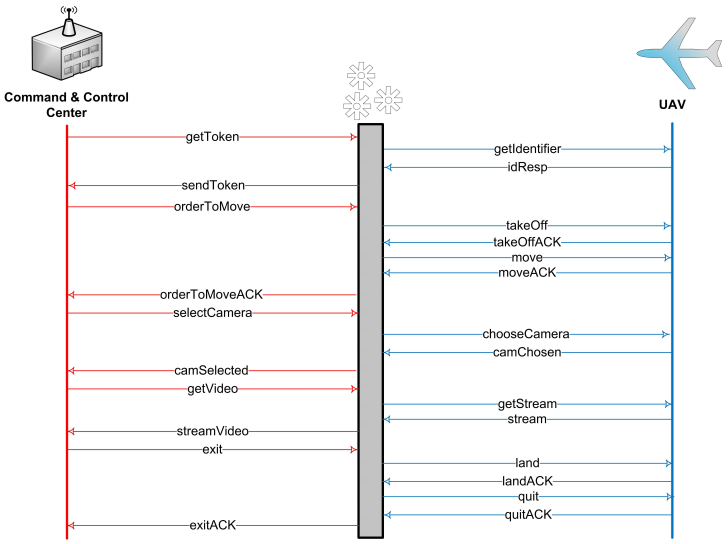


Fig. 2. Sequence diagram of the CONNECTed system

through the Mobius tool [26].
SAN models for the two networked systems (that is, the Command and Control Center and the UAV) and of the synthesised CONNECTor are generated. Here, we just show in Figure 3 the model of the CONNECTor, since our approach to dependability and performance enhancement acts only on it (in fact, the networked systems are assumed to be observable only through their interface and there is no possibility to make changes into their internal structure and operation).

In order to cover both performance and dependability aspects, the measures assessed in the evaluation are respectively latency and coverage. It is important to note that the measures assessed refer to two different and independent analyses, detailed in the following.

Analysis of Performance. *Latency* is evaluated as a performance indicator and is measured from when the Command and Control Center sends one of the possible orders to move (`orderToMove`) to when it receives an acknowledgement back (`orderToMoveACK`). Given the initial parameters values characterising the timing aspects of the elements involved in the CONNECTed system model (namely, the time to communicate between the networked systems and the CONNECTor), and the latency requirement stated beforehand, the analysis reveals that the CONNECTor does not satisfy this last. This implies that some enhancement is necessary, and this is the point where our approach plays its part. In this case study, we limit our identification of the appropriate mechanism to select to those listed in Table 1.

In the case of latency requirement not satisfied, Probing and the Parallel Retry mechanisms are the candidate mechanisms to improve the CONNECTed

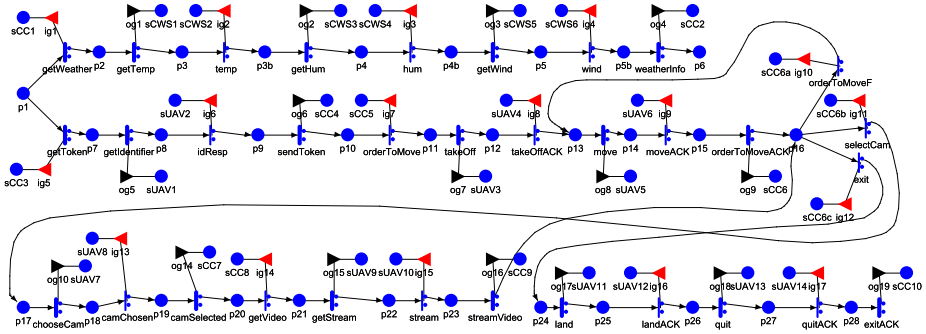


Fig. 3. SAN of the CONNECTor

system with respect to timing constraints. In the presence of limited availability of channels in exclusive usage by the CONNECTed system under analysis, preference goes to the Probing mechanism, which we applied to our example. Then, the second phase is to select the possible model element(s) to enhance with Probing. Following the steps delineated in Section 3.2, the element for which the mechanism brings the highest benefit in decreasing the latency metric is the `move` activity.

Figure 4 shows the results obtained considering the measure of interest (on the y axis expressed in time units) at varying of the rate of the distribution between 0.1 to 1 (on the x axis). The plots in Figure 4 show the latency results distinguishing the first movement, which includes the time needed for the take off operations, from all the other movements that are performed during the flight. As expected, the latency during the first phase, which includes the take off, is greater than the flight phase. Figure 4 also shows the results obtained including the use of the *Probing mechanism* [21] in the model of the CONNECTed system, thus allowing the enhancement of the requirements of latency in both the phases.

Analysis of Dependability. The second analysis aims to assess aspects of dependability of the CONNECTed system through the *Coverage*, defined as the percentage of stream video the Command and Control Center correctly receives from the UAV, with respect to the number of video requests made.

Figure 5 shows the trend of coverage (on the y axis) at increasing values of streams requests from C2-A (on the x axis). Moreover, the coverage threshold is shown in the figure at value of 0.75, as specified in the requirement. The figure shows that the analysis results obtained considering the basic model of the CONNECTed system satisfy the requirement only when the number of requests is at most 13.

As for the performance analysis, Table 1 needs to be examined to find the proper mapping between the mechanism and the characterising aspects when the coverage requirement is not met. Here, timing constraints are rather soft, while correctness is mainly relevant. Therefore, the viable candidate mechanisms

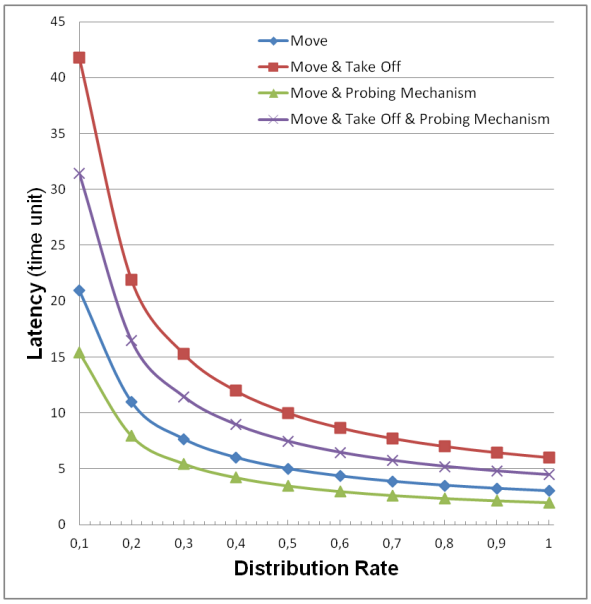


Fig. 4. Latency assessment at varying of the rate of the distribution

are Retry and Majority Voting. For the sake of saving in additional resources to employ, the Retry mechanism is selected. By applying the procedure that selects the model element(s) that needs to be enhanced, we find that `getStream` is the activity that leads to the greatest improvement with the Retry mechanism. This activity models two communication channels between the CONNECTor and the UAV. The results in Figure 5 confirm that Retry applied to that activity provides an improvement on the measure of coverage with respect to the original model. The enhanced CONNECTor meets the requirement up to a maximum of 18 requests.

In order to verify a further possible improvement, analysis were performed by applying the Retry mechanism on a further activity, `stream`, that is the second critical activity obtained through the selection procedure.

It is possible to note that the use of this dependability mechanism on two different activities allows to fully meet the requirement of coverage.

For the sake of argument, in Figure 6 is shown a portion of the CONNECTor model where the `getStream` activity has been replaced by the Retry Mechanism between the places `p21` and `p22`. It can be noticed that the mechanism can be directly used in the model to replace any activity that moves tokens between two places in the specification of the CONNECTor. In fact, as explained in [21], the mechanisms are developed according to three basic rules that allow to simplify the automated procedure for embedding the mechanism in the specification of the synthesised CONNECTor: (i) each model has an initial place, `s0`, whose tokens enable the first activity of the model; (ii) each model has a final place, `s1`, which

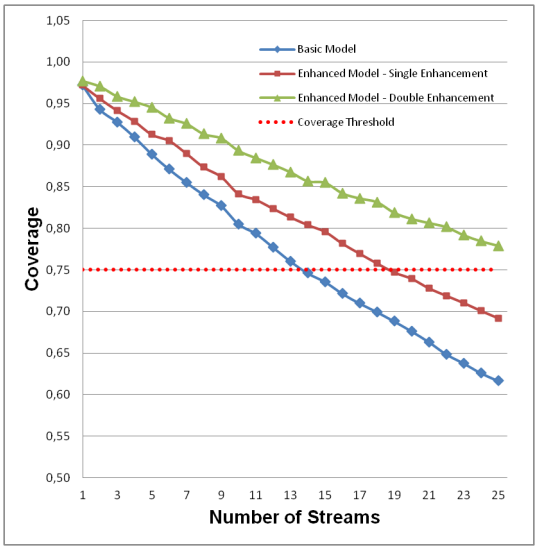


Fig. 5. Coverage assessment at varying of number of streams request

contains tokens whenever the last activity of the model completes; (iii) the overall number of tokens in s1 is always less or equal to the number of tokens in `s0`.

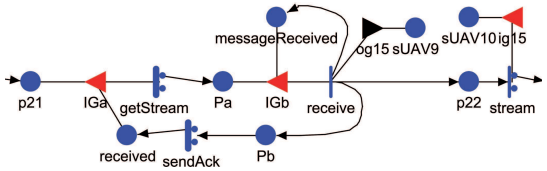


Fig. 6. Portion of the CONNECTor model, replaced by the Retry mechanism

5 Related work

A number of solutions have been presented in the literature to deal with critical applications requiring degrees of fault tolerance or efficiency, including classical architectural mechanisms such as N-Version Programming [5], Recovery Block [22], Cooperative Backup [14] and dynamic mechanisms such as dynamic function allocation [17]. The set of dependability mechanisms considered in this paper are fully inspired to them. There are also studies where the most popular fault tolerance solutions are discussed and classified according to a variety of key characteristics, e.g. the main direct benefits of the mechanism in terms of improved system resilience or assurance of key system properties and the types of system

within which this mechanism can be applied [23, 24], with the intention to assist in the selection and configuration of mechanisms at design-time and run-time. A review of several technologies supporting dynamic selection of components and services, such as Multi-Agent Systems, GRID computing, Web Services, is also included in [23]. Here, we are not tight to any specific technology, but rather we are focusing on generally applicable solutions in the specific context of dependable networks interoperability in heterogeneous environments.

Solutions for automatic addition of fault tolerance to fault intolerant programs have been also proposed, both resorting to a set of symbolic heuristics for synthesizing fault-tolerant distributed programs such as in [7] and by developing formal algorithms working in presence of a specific set of faults, a safety condition, and a reachability constraint as in [16], where it is also shown that the synthesis problem in the context of distributed programs is NP-complete in the state space of the given intolerant program. Further investigations on theoretical aspects and the development of a software framework for the synthesis of fault-tolerant programs are in [9].

Also, in the context of (*reflective* systems, i.e. systems having the property of enabling observation and control of its own structure and behavior from outside itself, independent development of fault tolerance libraries and applications has been carried on as a means to enhance the flexibility of dependable systems by providing properties like: ease of use and transparency of mechanisms for the application programmer, seamless reuse and extension of both functional and non-functional software and composition of mechanisms [25]. Moving to autonomic systems [13], their control loop requires assistance by a methodology and supporting technique to select and apply countermeasures to faults experienced during the system lifetime, possibly resorting to a library of fault tolerance patterns. However, to the best of our knowledge, there isn't in the literature a general approach developed to select from a library of fault tolerance mechanisms, whose efficacy to meet specified dependability and performance requirements is checked through model-based analysis. Our work have been tailored to explore this direction, .

6 Conclusions

We have illustrated an approach to automate the selection of a suitable dependability mechanism. This has been discussed in the context of networks of interoperable systems, as targeted by the EU project CONNECT. In particular, the approach targets generic mechanisms based on the synthesis of mediating software bridges (CONNECTors) that allow interoperability among heterogeneous devices. In addition, we have investigated a generic method for identifying elements in the CONNECTor that must be reinforced with the selected dependability mechanism in order to improve performance– and dependability–related metrics. The approach has been demonstrated through a case study based on the GMES

European Programme. Specifically, a scenario describing the management of forest-fire at the border between two different countries has been considered. In the considered scenario, resources provided by both countries provide the same functionalities but use different protocols and therefore a CONNECTor is necessary to allow interoperation.

The work presented in this paper sets the basis for an automated approach to select a dependability mechanism and how to apply them on a model of the system in order to meet given requirements. Further investigations would be valuable to carry on, especially in the directions of: i) detailing the ontological matching approach by defining exemplary application-dependent ontologies for widely used networked applications; ii) making more efficient the strategy devoted to select the elements where the dependability mechanism has to be applied by investigating methods to predict, with a satisfactory level of accuracy, the minimum set of elements that would be necessary for requirement satisfaction within the just one additional analysis phase; iii) implement these methods in the dependability assessment tool (the DEPER unit under development in the CONNECT project).

Acknowledgements

This work is partially supported by the EU FP7 Project CONNECT (FP7–231167).

References

1. Ieee std 610.12-1990 (n.d.). ieee standard glossary of software engineering terminology (1990), http://ieeexplore.ieee.org/
2. Ann T. Tai, J.F.M., Avizienis, A.: Software Performability: from concepts to Applications. Kluwer Academic Publisher (1996)
3. Asare, P., Cong, D., Vattam, S.G., Kim, B., King, A., Sokolsky, O., Lee, I., Lin, S., Mullen-Fortino, M.: The medical device dongle: an open-source standards-based platform for interoperable medical device connectivity. In: Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium. pp. 667–672. IHI '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2110363.2110438
4. Avizienis, A., Kelly, J.P.J.: Fault tolerance by design diversity: Concepts and experiments. Computer 17(8), 67–80 (Aug 1984), http://dx.doi.org/10.1109/MC.1984.1659219
5. Avizienis, A.: The n-version approach to fault-tolerant software. IEEE Transactions on Software Engineering SE-11(12), 1491–1501 (December 1985)
6. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secur. Comput. 1(1), 11–33 (Jan 2004), http://dx.doi.org/10.1109/TDSC.2004.2
7. Bonakdarpour, B., Kulkarni, S.S.: Sycraft: A tool for synthesizing distributed fault-tolerant programs. In: CONCUR 2008: International Conference on Concurrency Theory. pp. 167–171. Springer Verlag (2008)

8. CONNECT Consortium: Deliverable 6.3 – Experiment scenarios, prototypes and report Iteration 2 (2012)
9. Ebnenasir, A.: Automatic synthesis of fault-tolerance (2005), phD Dissertation, Michigan State University
10. Food, U., (FDA), D.A.: Fda safety communication: Preventing surgical fires (October 2011), http://www.fda.gov/MedicalDevices/Safety/AlertsandNotices/ucm275189.htm
11. France, R., Rumpe, B.: Model-driven development of complex software: A research roadmap. In: 2007 Future of Software Engineering. pp. 37–54. FOSE '07, IEEE Computer Society, Washington, DC, USA (2007), http://dx.doi.org/10.1109/FOSE.2007.14
12. Gruber, T.R., Olsen, G.R.: An ontology for engineering mathematics. In: Doyle, J., Sandewall, E., Torasso, P. (eds.) KR. pp. 258–269. Morgan Kaufmann (1994)
13. Huebscher, M.C., McCann, J.A.: A survey of autonomic computingdegrees, models, and applications. ACM Computing Surveys 40(3), 237 – 254 (2008)
14. Killijian, M.O., Courtes, L., Powell, D.: A survey of cooperative backup mechanisms (October 2006), lAAS Technical Report 06472
15. King, A., Roederer, A., Chen, S., Stevens, N., Asare, P., Sokolsky, O., Lee, I., Mullen-Fortino, M., Park, S.: Demo of the generic smart alarm: a framework for the design, analysis, and implementation of smart alarms and other clinical decision support systems. In: Wireless Health 2010. pp. 210–211. WH '10, ACM, New York, NY, USA (2010), http://doi.acm.org/10.1145/1921081.1921116
16. Kulkarni, S.S., Arora, A.: Automating the addition of fault-tolerance. In: 6th Formal Techniques in Real-Time and Fault-Tolerant Systems. p. 8293 (2000)
17. M, T., M., H.: The temporal dimension of dynamic function allocation. In: Proceeding of 11th European Conference on Cognitive Ergonomics (ECCE). pp. 283–291 (2002)
18. Masci, P., Chiaradonna, S., Giandomenico, F.D.: Dependability analysis of diffusion protocols in wireless networks with heterogeneous node capabilities. In: Proceedings of the 2010 European Dependable Computing Conference. pp. 145–154. EDCC '10, IEEE Computer Society, Washington, DC, USA (2010), http://dx.doi.org/10.1109/EDCC.2010.26
19. Masci, P., Furniss, D., Curzon, P., Harrison, M., Blandford, A.: Supporting field investigators with pvs: A case study in the healthcare domain. In: Avgeriou, P. (ed.) Software Engineering for Resilient Systems, Lecture Notes in Computer Science, vol. 7527, pp. 150–164. Springer Berlin Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-33176-3_11
20. Masci, P., Martinucci, M., Giandomenico, F.D.: Towards automated dependability analysis of dynamically connected systems. In: Proceedings of the 2011 Tenth International Symposium on Autonomous Decentralized Systems. pp. 139–146. ISADS '11, IEEE Computer Society, Washington, DC, USA (2011), http://dx.doi.org/10.1109/ISADS.2011.23
21. Masci, P., Nostro, N., Di Giandomenico, F.: On enabling dependability assurance in heterogeneous networks through automated model-based analysis. In: Proceedings of the Third international conference on Software engineering for resilient systems. pp. 78–92. SERENE'11, Springer-Verlag, Berlin, Heidelberg (2011), http://dl.acm.org/citation.cfm?id=2045537.2045548
22. Randell, B.: System structure for software fault tolerance. IEEE Transactions on Software Engineering SE-1(2), 221–232 (1975)

23. ReSIST Consortium: EU project ReSIST: Resilience for Survivability in IST. Deliverable D11: Support for Resilience-Explicit Computing - first edition. Tech. rep. (2007), http://www.resist-noe.org/
24. ReSIST Consortium: EU project ReSIST: Resilience for Survivability in IST. Deliverable D33: Resilience-explicit Computing - final. Tech. rep. (2008), http://www.resist-noe.org/
25. Ruiz, J., Killijian, M.O., Fabre, J.C., Thvenod-Fosse, P.: Reflective fault-tolerant systems: from experience to challenges. IEEE Transactions on Computers 52(2), 237 – 254 (2003)
26. Sanders, W.H., Meyer, J.F.: Stochastic Activity Networks: formal definitions and concepts pp. 315–343 (2002)
27. Spalazzese, R., Inverardi, P.: Mediating connector patterns for components interoperability. In: Proceedings of the 4th European conference on Software architecture. pp. 335–343. ECSA'10, Springer-Verlag, Berlin, Heidelberg (2010), http://dl.acm.org/citation.cfm?id=1887899.1887928
28. Stevens, N., Giannareas, A.R., Kern, V., Viesca, A., Fortino-Mullen, M., King, A., Lee, I.: Smart alarms: multivariate medical alarm integration for post cabg surgery patients. In: Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium. pp. 533–542. IHI '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2110363.2110423